

CIS GitHub Benchmark

v1.0.0 - 12-28-2022

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	6
Intended Audience	6
Consensus Guidance	7
Typographical Conventions	8
Recommendation Definitions	9
Title	9
Assessment Status	9
Automated	9
Manual	9
Profile	9
Description	9
Rationale Statement	9
Impact Statement	10
Audit Procedure	10
Remediation Procedure	10
Default Value	10
References	10
CIS Critical Security Controls® (CIS Controls®)	10
Additional Information	10
Profile Definitions	11
Acknowledgements	12
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)	29
1.1.9 Ensure all checks have passed before merging new code (Manual)	31
1.1.10 Ensure open Git branches are up to date before they can be merged into code base (Manual)	33

1.1.11 Ensure all open comments are resolved before allowing code change merging (Manual)	35
1.1.12 Ensure verification of signed commits for new changes before merging (Manual)	37
1.1.13 Ensure linear history is required (Manual)	39
1.1.14 Ensure branch protection rules are enforced for administrators (Manual)	41
1.1.15 Ensure pushing or merging of new code is restricted to specific individuals or teams (Manual)	43
1.1.16 Ensure force push code to branches is denied (Manual)	45
1.1.17 Ensure branch deletions are denied (Manual)	47
1.1.18 Ensure any merging of code is automatically scanned for risks (Manual)	49
1.1.19 Ensure any changes to branch protection rules are audited (Manual)	51
1.1.20 Ensure branch protection is enforced on the default branch (Manual)	53
1.2 Repository Management	55
1.2.1 Ensure all public repositories contain a SECURITY.md file (Manual)	56
1.2.2 Ensure repository creation is limited to specific members (Manual)	58
1.2.3 Ensure repository deletion is limited to specific users (Manual)	60
1.2.4 Ensure issue deletion is limited to specific users (Manual)	63
1.2.5 Ensure all copies (forks) of code are tracked and accounted for (Manual)	65
1.2.6 Ensure all code projects are tracked for changes in visibility status (Manual)	67
1.2.7 Ensure inactive repositories are reviewed and archived periodically (Manual)	69
1.3 Contribution Access	71
1.3.1 Ensure inactive users are reviewed and removed periodically (Manual)	72
1.3.2 Ensure team creation is limited to specific members (Manual)	74
1.3.3 Ensure minimum number of administrators are set for the organization (Manual)	76
1.3.4 Ensure Multi-Factor Authentication (MFA) is required for contributors of new code (Manual)	78
1.3.5 Ensure the organization is requiring members to use Multi-Factor Authentication (MFA) (Manual)	80
1.3.6 Ensure new members are required to be invited using company-approved email (Manual)	82
1.3.7 Ensure two administrators are set for each repository (Manual)	84
1.3.8 Ensure strict base permissions are set for repositories (Manual)	86
1.3.9 Ensure an organization's identity is confirmed with a "Verified" badge (Manual)	88
1.3.10 Ensure Source Code Management (SCM) email notifications are restricted to verified domains (Manual)	90
1.3.11 Ensure an organization provides SSH certificates (Manual)	92
1.3.12 Ensure Git access is limited based on IP addresses (Manual)	94
1.3.13 Ensure anomalous code behavior is tracked (Manual)	96
1.4 Third-Party	97
1.4.1 Ensure administrator approval is required for every installed application (Manual)	98
1.4.2 Ensure stale applications are reviewed and inactive ones are removed (Manual)	100
1.4.3 Ensure the access granted to each installed application is limited to the least privilege needed (Manual)	102
1.4.4 Ensure only secured webhooks are used (Manual)	104
1.5 Code Risks	106
1.5.1 Ensure scanners are in place to identify and prevent sensitive data in code (Manual)	107
1.5.2 Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions (Manual)	109
1.5.3 Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions (Manual)	111
1.5.4 Ensure scanners are in place for code vulnerabilities (Manual)	113
1.5.5 Ensure scanners are in place for open-source vulnerabilities in used packages (Manual)	115
1.5.6 Ensure scanners are in place for open-source license issues in used packages (Manual)	117
2 Build Pipelines	118
2.1 Build Environment	119
2.1.1 Ensure each pipeline has a single responsibility (Manual)	120
2.1.2 Ensure all aspects of the pipeline infrastructure and configuration are immutable (Manual)	122
2.1.3 Ensure the build environment is logged (Manual)	124
2.1.4 Ensure the creation of the build environment is automated (Manual)	125

2.1.5 Ensure access to build environments is limited (Manual).....	126
2.1.6 Ensure users must authenticate to access the build environment (Manual)	128
2.1.7 Ensure build secrets are limited to the minimal necessary scope (Manual).....	129
2.1.8 Ensure the build infrastructure is automatically scanned for vulnerabilities (Manual)	131
2.1.9 Ensure default passwords are not used (Manual).....	132
2.1.10 Ensure webhooks of the build environment are secured (Manual)	133
2.1.11 Ensure minimum number of administrators are set for the build environment (Manual).....	134
2.2 Build Worker	135
2.2.1 Ensure build workers are single-used (Manual)	136
2.2.2 Ensure build worker environments and commands are passed and not pulled (Manual).....	137
2.2.3 Ensure the duties of each build worker are segregated (Manual).....	138
2.2.4 Ensure build workers have minimal network connectivity (Manual)	139
2.2.5 Ensure run-time security is enforced for build workers (Manual)	141
2.2.6 Ensure build workers are automatically scanned for vulnerabilities (Manual).....	142
2.2.7 Ensure build workers' deployment configuration is stored in a version control platform (Manual) ...	143
2.2.8 Ensure resource consumption of build workers is monitored (Manual).....	145
2.3 Pipeline Instructions	146
2.3.1 Ensure all build steps are defined as code (Manual).....	147
2.3.2 Ensure steps have clearly defined build stage input and output (Manual)	148
2.3.3 Ensure output is written to a separate, secured storage repository (Manual)	149
2.3.4 Ensure changes to pipeline files are tracked and reviewed (Manual)	150
2.3.5 Ensure access to build process triggering is minimized (Manual).....	151
2.3.6 Ensure pipelines are automatically scanned for misconfigurations (Manual).....	152
2.3.7 Ensure pipelines are automatically scanned for vulnerabilities (Manual)	153
2.3.8 Ensure scanners are in place to identify and prevent sensitive data in pipeline files (Automated) ..	154
2.4 Pipeline Integrity	155
2.4.1 Ensure all artifacts on all releases are signed (Manual).....	156
2.4.2 Ensure all external dependencies used in the build process are locked (Manual).....	157
2.4.3 Ensure dependencies are validated before being used (Manual)	159
2.4.4 Ensure the build pipeline creates reproducible artifacts (Manual).....	161
2.4.5 Ensure pipeline steps produce a Software Bill of Materials (SBOM) (Manual)	162
2.4.6 Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced (Manual).....	163
3 Dependencies	164
3.1 Third-Party Packages	165
3.1.1 Ensure third-party artifacts and open-source libraries are verified (Manual).....	166
3.1.2 Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers (Manual)	168
3.1.3 Ensure signed metadata of the build process is required and verified (Manual).....	170
3.1.4 Ensure dependencies are monitored between open-source components (Manual)	172
3.1.5 Ensure trusted package managers and repositories are defined and prioritized (Manual)	173
3.1.6 Ensure a signed Software Bill of Materials (SBOM) of the code is supplied (Manual)	175
3.1.7 Ensure dependencies are pinned to a specific, verified version (Manual)	177
3.1.8 Ensure all packages used are more than 60 days old (Manual)	179
3.2 Validate Packages	181
3.2.1 Ensure an organization-wide dependency usage policy is enforced (Manual).....	182
3.2.2 Ensure packages are automatically scanned for known vulnerabilities (Manual)	183
3.2.3 Ensure packages are automatically scanned for license implications (Manual).....	184
3.2.4 Ensure packages are automatically scanned for ownership change (Manual)	186
4 Artifacts.....	187
4.1 Verification.....	188
4.1.1 Ensure all artifacts are signed by the build pipeline itself (Manual).....	189
4.1.2 Ensure artifacts are encrypted before distribution (Manual).....	192

4.1.3 Ensure only authorized platforms have decryption capabilities of artifacts (Manual).....	193
4.2 Access to Artifacts.....	195
4.2.1 Ensure the authority to certify artifacts is limited (Manual)	196
4.2.2 Ensure number of permitted users who may upload new artifacts is minimized (Manual).....	197
4.2.3 Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA) (Manual)...	198
4.2.4 Ensure user management of the package registry is not local (Manual).....	199
4.2.5 Ensure anonymous access to artifacts is revoked (Manual)	200
4.2.6 Ensure minimum number of administrators are set for the package registry (Manual)	202
4.3 Package Registries	204
4.3.1 Ensure all signed artifacts are validated upon uploading the package registry (Manual).....	205
4.3.2 Ensure all versions of an existing artifact have their signatures validated (Manual)	207
4.3.3 Ensure changes in package registry configuration are audited (Manual).....	209
4.3.4 Ensure webhooks of the repository are secured (Manual)	211
4.4 Origin Traceability.....	212
4.4.1 Ensure artifacts contain information about their origin (Manual)	213
5 Deployment.....	214
5.1 Deployment Configuration	215
5.1.1 Ensure deployment configuration files are separated from source code (Manual)	216
5.1.2 Ensure changes in deployment configuration are audited (Manual)	217
5.1.3 Ensure scanners are in place to identify and prevent sensitive data in deployment configuration (Manual)	218
5.1.4 Limit access to deployment configurations (Manual).....	219
5.1.5 Scan Infrastructure as Code (IaC) (Manual).....	221
5.1.6 Ensure deployment configuration manifests are verified (Manual).....	222
5.1.7 Ensure deployment configuration manifests are pinned to a specific, verified version (Manual)	223
5.2 Deployment Environment.....	225
5.2.1 Ensure deployments are automated (Manual)	226
5.2.2 Ensure the deployment environment is reproducible (Manual)	227
5.2.3 Ensure access to production environment is limited (Manual)	228
5.2.4 Ensure default passwords are not used (Manual).....	229
Appendix: Summary Table.....	230
Appendix: CIS Controls v7 IG 1 Mapped Recommendations	240
Appendix: CIS Controls v7 IG 2 Mapped Recommendations	242
Appendix: CIS Controls v7 IG 3 Mapped Recommendations	246
Appendix: CIS Controls v7 Unmapped Recommendations	251
Appendix: CIS Controls v8 IG 1 Mapped Recommendations	253
Appendix: CIS Controls v8 IG 2 Mapped Recommendations	255
Appendix: CIS Controls v8 IG 3 Mapped Recommendations	260
Appendix: CIS Controls v8 Unmapped Recommendations	266
Appendix: Change History	267

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

Resheet Kosef
Eylam Milner

Editor

Randall Mowen

Contributor

Phil White
Andrew Dannenberger
Lucas Aides
Marina Segal
Kari Byrd
Yossi Weizman
Erez Dasa
Michael Kotelnikov
Moshik Barak
Yakir Kadkoda
Mor Weinberger
Stephen Keller
Ori Zerah
Marina Segal CISA, CRISC, C|CISO
Ofir Shapira
Ronen Slavin
Alex Ilgayev
Matthew Reagan

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 through Github repository for every micro-service or application developed by an organization.

Remediation:

Upload existing code projects to a dedicated Github organization and repositories 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 a task management system 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 (ticket) in your organization's task management system. These tasks should also be linked to the code changes themselves in a way that is easy to follow: from code to task, and from task back to 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.			

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

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require a pull request before merging** and **Require approvals** are checked, and verify that **Required number of approvals before merging** is set to 2.

Remediation:

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

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you added the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Check **Require a pull request before merging** and **Require approvals**, and set **Required number of approvals before merging** to 2.
7. Click **Create** or **Save changes**.

Default Value:

0

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	18.8 <u>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.		●	●

1.1.4 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:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require a pull request before merging** is checked, and verify that **Dismiss stale pull request approvals when new commits are pushed** is checked.

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:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.

4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you added the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require pull request reviews before merging** and then **Dismiss stale pull request approvals when new commits are pushed**.
7. Click **Create** or **Save changes**.

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:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Verify that **Require a pull request before merging** and **Restrict who can dismiss pull request reviews** is checked.
6. Verify that no users and teams are specified except for organization and repository admins. If it is obligatory, verify that the users or teams specified were carefully selected to be trusted with the ability to dismiss code change reviews.

Remediation:

For each code repository in use, perform the next steps to restrict dismissal of code changes reviews unless it is necessary:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you added the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require pull request reviews before merging** and **Restrict who can dismiss pull request reviews**.
7. Do not add any user or team unless it is obligatory. If it is obligatory, carefully select the users or teams whom you trust with the ability to dismiss code change reviews.
8. Click **Create** or **Save changes**.

Default Value:

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

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>14.7 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.			

1.1.6 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 code repository, verify that a file named CODEOWNERS exists in the root, docs/, or .github/ directory of the repository.

In the CODEOWNERS file, verify that the users specified are users you trust.

Remediation:

In every code repository create a CODEOWNERS file in the root, docs/, or .github/ directory of the repository.

In the file, specify codeowners for the .github/workflows/ directory atleast. Specify organization members you trust. For example:

```
.github/workflows/ @user1 @user2
```

Default Value:

None

References:

1. <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-code-owners>

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.			

1.1.7 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:

For every code repository in use, perform the following steps to verify that code owners are required to review all code change proposals relevant to areas they own before code merge:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require a pull request before merging** and **Require review from Code Owners** are checked.

Remediation:

For every code repository in use, perform the following steps to require code owners' approvals for each change proposal related to code they own:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.

3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require pull request reviews before merging** and **Require review from Code Owners**.
7. Click **Create** or **Save changes**.

Default Value:

Code owners are not required to review changes by default.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.9 Train Developers in Application Security Concepts and Secure Coding</u> Ensure that all software development personnel receive training in writing secure code for their specific development environment and responsibilities. Training can include general security principles and application security standard practices. Conduct training at least annually and design in a way to promote security within the development team, and build a culture of security among the developers.			
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 code repository in use, verify that all existing Git branches are active or have yet to be checked for inactivity by performing the next steps:

1. On GitHub.com, navigate to the main page of the repository.
2. Above the list of files, click **Branches**.
3. 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.
4. 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.

Remediation:

For each code repository in use, review existing Git branches and remove those which have not been active for a period of time by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Above the list of files, click **Branches**.

3. 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.
4. For each branch listed, either delete it by clicking the trash bin icon, or find the valid reason it still exists.

You can perform the next steps to prevent pull request branches from becoming stale branches:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click Settings.
3. Under "Pull Requests", select **Automatically delete head branches**.

Default Value:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.11 Leverage Vetted Modules or Services for Application Security Components</u> Leverage vetted modules or services for application security components, such as identity management, encryption, and auditing and logging. Using platform features in critical security functions will reduce developers' workload and minimize the likelihood of design or implementation errors. Modern operating systems provide effective mechanisms for identification, authentication, and authorization and make those mechanisms available to applications. Use only standardized, currently accepted, and extensively reviewed encryption algorithms. Operating systems also provide mechanisms to create and maintain secure audit logs.			
v7	<u>18.2 Ensure Explicit Error Checking is Performed for All In-house Developed Software</u> For in-house developed software, ensure that explicit error checking is performed and documented for all input, including for size, data type, and acceptable ranges or formats.			

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:

For each code repository in use, verify that status checks are required to pass before allowing any code change proposal merge by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require status checks to pass before merging** is checked.

Remediation:

For each code repository in use, require all status checks to pass before permitting a merge of new code by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", check if there is a rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.

5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require status checks to pass before merging**.
7. Click **Create** or **Save changes**.

Default Value:

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

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.			

1.1.10 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 code repository in use, verify that open branches must be updated before merging is permitted by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require status checks to pass before merging** and **Require branches to be up to date before merging** are checked.

Remediation:

For each code repository in use, enforce a policy to only allow merging open branches if they are current with the latest change from their original repository by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.

4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require status checks to pass before merging** and **Require branches to be up to date before merging**.
7. Click **Create** or **Save changes**.

Default Value:

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

References:

1. <https://github.blog/changelog/2022-02-03-more-ways-to-keep-your-pull-request-branch-up-to-date/>

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.			

1.1.11 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:

For every code repository in use, verify that each merged code change does not contain open, unattended comments by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require conversation resolution before merging** is checked.

Remediation:

For each code repository in use, require open comments to be resolved before the relevant code change can be merged by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.

5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require conversation resolution before merging**.
7. Click **Create** or **Save changes**.

Default Value:

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

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:

Ensure only signed commits can be merged for every branch, especially the main branch, via branch protection rules by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require signed commits** is checked.

Remediation:

For each repository in use, enforce the branch protection rule of requiring signed commits, and make sure only signed commits are capable of merging by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require signed commits**.

7. Click **Create** or **Save changes**.

References:

1. <https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits>

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 code repository in use, perform the following steps to verify that linear history is required and/or that only squash merge and rebase merge are allowed:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Require linear history** is checked.

Remediation:

For every code repository in use, perform the following steps to require linear history and/or allow only rebase merge and squash merge:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Select **Require linear history**.

7. Click **Create** or **Save changes**.

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:

For every code repository in use, validate branch protection rules also apply to administrator accounts by performing the next steps:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Do not allow bypassing the above settings** is checked.

Remediation:

For every code repository in use, enforce branch protection rules on administrators as well, by performing the following:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.

6. Select **Do not allow bypassing the above settings**.
7. Click **Create** or **Save changes**.

Default Value:

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

References:

1. <https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/defining-the-mergeability-of-pull-requests/managing-a-branch-protection-rule>

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 GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Restrict who can push to matching branches** is checked and that only trusted and responsible users and teams are selected.

Remediation:

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

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
4. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
5. Select **Restrict who can push to matching branches** and choose trusted and responsible users and teams who will have the permission to do so.

6. Click **Create** or **Save changes**.

References:

1. <https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/defining-the-mergeability-of-pull-requests/managing-a-branch-protection-rule>

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.			

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:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Allow force pushes** is not checked.

Remediation:

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

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Uncheck **Allow force pushes**.
7. Click **Create** or **Save changes**.

References:

1. <https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/defining-the-mergeability-of-pull-requests/managing-a-branch-protection-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.			

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:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, you are not compliant.
5. Ensure that **Allow deletions** is not checked.

Remediation:

For each repository that is being used, block the option to delete protected branches by performing the following:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", verify that there is at least one rule for your main branch. If there is, click **Edit** to its right. If there isn't, click **Add rule**.
5. If you add the rule, under "Branch name pattern", type the branch name or pattern you want to protect.
6. Uncheck **Allow deletions**.
7. Click **Create** or **Save changes**.

References:

1. <https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/defining-the-mergeability-of-pull-requests/managing-a-branch-protection-rule>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.4 Enforce Data Retention Retain data according to the enterprise's data management process. Data retention must include both minimum and maximum timelines.			
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.			

1.1.18 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 repository in use, ensure that every pull request must be scanned for risks by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.
3. Ensure that at least one workflow is configured to run like that:

```
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
```

and that it has a step that runs code scanning on the code.

Remediation:

For every repository in use, enforce risk scanning on every pull request by performing the following:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.
3. If the repository already has at least one workflow set up and running, click **New workflow** and go to step 5. If there are currently no workflows configured for the repository, go to the next step.
4. Scroll down to the "Security" category and click **Configure** under the workflow you want to configure or click **View all** to see all available security workflows.
5. On the right pane of the workflow page, click **Documentation** and follow the on-screen instructions to tailor the workflow to your needs.

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.			

1.1.19 *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 changes in branch protection rules are audited by performing the following regularly:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Archives" section of the sidebar, click **Logs**, then click **Audit log**.
4. Use the action qualifier in your query and look for **protected_branch** category. 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. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Archives" section of the sidebar, click **Logs**, then click **Audit log**.
4. Use the action qualifier in your query and look for **protected_branch** category. Ensure every action is reasonable and secure and investigate if not.

References:

1. <https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/defining-the-mergeability-of-pull-requests/about-protected-branches>

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	6.2 <u>Activate audit logging</u> Ensure that local logging has been enabled on all systems and networking devices.			

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:

Perform the following to ensure branch protection is enforced on the main branch:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Under **Branch protection rules**, verify that there is a rule applied to the "main" or default branch.

Remediation:

Perform the following to enforce branch protection on the main branch:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Code and automation" section of the sidebar, click **Branches**.
4. Next to "Branch protection rules", click **Add rule**.
5. Under "Branch name pattern", type the branch name or pattern you want to protect. Ensure it applies to the main branch name.
6. Configure policies you want.
7. Click Create.

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.			

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:

1. On GitHub.com, navigate to the main page of the repository.
2. Under the repository name, click **Security**.
3. Verify that **Security policy** is enabled.

Remediation:

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

1. On GitHub.com, navigate to the main page of the repository.
2. Under the repository name, click **Security**.
3. In the left sidebar, click **Security policy**.
4. Click **Start setup**.
5. In the new SECURITY.md file, add information about supported versions of your project and how to report a vulnerability.
6. At the bottom of the page, type a commit message.
7. Below the commit message fields, choose to create a new branch for your commit and then create a pull request.
8. Click **Propose file change**.

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:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Access" section of the sidebar, click **Member privileges**.
4. Under "Repository creation", ensure that **Public** and **Private** are not checked.
This means only owners are able to create repositories.

Remediation:

Restrict repository creation to trusted users and teams only by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Access" section of the sidebar, click **Member privileges**.
4. Under "Repository creation", unselect both options - **Public** and **Private**.
5. Click **Save**.

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.			

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:

If Your organizations > Settings > Access > Member privileges > Allow members to delete or transfer repositories for this organization is selected, verify that every admin member is trusted by you:

1. In every repository, on GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings** and then in the "Access" section of the sidebar, click **Collaborators & teams**.
3. Under "Manage access" use the dropdown **Role** menu to filter and search for admin members. Verify that there are only two of them and that they are trusted and qualified.

If it is not selected, verify that every organization owner is trusted by you:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Click the name of your organization and under your organization name, click **People**.
3. You will see a list of the people in your organization. Click Role and select Owners. Verify that there are only a few of them and that they are trusted and qualified.

In any case, only members with administrator or organization owner privileges can delete repositories, regardless of your setting.

Remediation:

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

If Your organizations > Settings > Access > Member privileges > Allow members to delete or transfer repositories for this organization is selected, allow only trusted members to have admin privileges:

1. In every repository, on GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings** and then in the "Access" section of the sidebar, click **Collaborators & teams**.
3. Under "Manage access" use the dropdown **Role** menu to filter and search for admin members. Change their role by clicking the **Role** dropdown next to the username until there are only two trusted and qualified users with admin privileges.

If it is not selected, allow only trusted users to become an organization owner:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Click the name of your organization and under your organization name, click **People**.
3. You will see a list of the people in your organization. Click Role and select Owners. Check every member you want to change permissions to. Above the list of members, use the drop-down menu and click Change role and choose Member > change role. Do that until there are only a few trusted and qualified users with organization owner privileges.

In any case, only members with administrator or organization owner privileges can delete repositories, regardless of your setting.

Default Value:

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

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.			

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:

If Your organizations > Settings > Access > Member privileges > Allow members to delete issues for this organization is selected, verify that every admin member is trusted by you:

1. In every repository, on GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings** and then in the "Access" section of the sidebar, click **Collaborators & teams**.
3. Under "Manage access" use the dropdown **Role** menu to filter and search for admin members. Verify that there are only two of them and that they are trusted and qualified.

If it is not selected, verify that every organization owner is trusted by you:

1. In the top right corner of GitHub.com, click your profile photo, then click Your organizations.
2. Click the name of your organization and under your organization name, click **People**.
3. You will see a list of the people in your organization. Click **Role** and select **Owners**. Verify that there are only a few of them and that they are trusted and qualified.

In any case, only members with administrator or organization owner privileges can delete issues, regardless of your setting.

Remediation:

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

If Your organizations > Settings > Access > Member privileges > Allow members to delete issues for this organization is selected, allow only trusted members to have admin privileges:

1. In every repository, on GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings** and then in the "Access" section of the sidebar, click **Collaborators & teams**.
3. Under "Manage access" use the dropdown **Role** menu to filter and search for admin members. Change their role by clicking the **Role** dropdown next to the username until there are only two trusted and qualified users with admin privileges.

If it is not selected, allow only trusted users to become an organization owner:

1. In the top right corner of GitHub.com, click your profile photo, then click Your organizations.
2. Click the name of your organization and under your organization name, click **People**.
3. You will see a list of the people in your organization. Click **Role** and select **Owners**. Check every member you want to change permissions to. Above the list of members, use the drop-down menu and click **Change role** and choose Member > change role. Do that until there are only a few trusted and qualified users with organization owner privileges.

In any case, only members with administrator or organization owner privileges can delete issues, regardless of your setting.

Default Value:

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

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.			

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:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.
3. In the left sidebar, click **Forks**.
4. Examine the forks listed there.

Remediation:

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

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.
3. In the left sidebar, click **Forks**.
4. Examine the forks listed there.

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.			

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 tracked and investigated, by performing the following regularly:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Archives" section of the sidebar, click **Logs**, then click **Audit log**.
4. Use the **repo** qualifier in your query and look for **access** category. Ensure every change is reasonable and secure and is investigated if it is not.

Remediation:

Track every change in project visibility and investigate if suspicious behavior occurs, by performing the following regularly:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Archives" section of the sidebar, click **Logs**, then click **Audit log**.
4. Use the **repo** qualifier in your query and look for **access** category. Ensure every change is reasonable and secure and investigate if it is not.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.14 Log Sensitive Data Access Log sensitive data access, including modification and disposal.			
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.			

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 repositories in the organization are active, and those that are not reviewed or archived:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Click on your organization name and then on **repositories**.
3. Ensure every repository listed has been active in the last 3 to 6 months. If it's not, then ensure it is archived or reviewed regularly.

Remediation:

Perform the following to review all inactive repositories and archive them periodically:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Click on your organization name and then on **repositories**.
3. Ensure every repository listed has been active in the last 3 to 6 months. Every repository that isn't active you should either review or archive by performing the next steps:
 1. On GitHub.com, navigate to the main page of the repository.
 2. Under your repository name, click **Settings**.
 3. Under "Danger Zone", click **Archive this repository**.
 4. Read the warnings.

5. Type the name of the repository you want to archive.
6. Click **I understand the consequences, archive this repository.**

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:

If you have GitHub AE, verify that all user accounts are active by performing the following:

1. From an administrative account on GitHub AE, in the upper-right corner of any page, click the rocket icon.
2. If you're not already on the "Site admin" page, in the upper-left corner, click **Site admin**.
3. In the left sidebar, click **Dormant users**.
4. Verify that the list is empty.

If you have GitHub Enterprise Cloud, perform the following:

1. In the top-right corner of GitHub.com, click your profile photo, then click **Your enterprises**.
2. In the list of enterprises, click the enterprise you want to view.
3. In the enterprise account sidebar, click **Compliance**.
4. To download your Dormant Users (beta) report as a CSV file, under "Other", click **Download**.
5. Verify that there are no users listed.

Remediation:

If you have GitHub AE, perform the following to review inactive user accounts and remove them:

1. From an administrative account on GitHub AE, in the upper-right corner of any page, click the rocket icon.

2. If you're not already on the "Site admin" page, in the upper-left corner, click **Site admin**.
3. In the left sidebar, click **Dormant users**.
4. Find the users listed there under **Your organizations** > your organization > **People** and select them.
5. Click **Remove from organization** and **Remove members**.

If you have GitHub Enterprise Cloud, perform the following:

1. In the top-right corner of GitHub.com, click your profile photo, then click **Your enterprises**.
2. In the list of enterprises, click the enterprise you want to view.
3. In the enterprise account sidebar, click **Compliance**.
4. To download your Dormant Users (beta) report as a CSV file, under "Other", click **Download**.
5. Find the users listed in the file under **Your organizations** > your organization > **People** and select them.
6. Click **Remove from organization** and **Remove members**.

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 team 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 team creation is limited to specific, trusted users by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Access" section of the sidebar, click **Member privileges**.
4. Under "Team creation rules", verify that **Allow members to create teams** is not selected.

Remediation:

For every organization, limit team creation to specific, trusted users by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Access" section of the sidebar, click **Member privileges**.

4. Under "Team creation rules", deselect **Allow members to create teams**.
5. Click **Save**.

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 organization by performing the following:

1. In the top right corner of GitHub, click your profile photo, then click **Your profile**.
2. On the left side of your profile page, under "Organizations", click the icon for your organization.
3. Under your organization name, click **People**.
4. In the Role drop-down, choose **Owners**.
5. If there are minimum number of members in the list, you are compliant.

Remediation:

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

1. In the top right corner of GitHub, click your profile photo, then click **Your profile**.
2. On the left side of your profile page, under "Organizations", click the icon for your organization.
3. Under your organization name, click **People**.
4. In the Role drop-down, choose **Owners**.
5. Select the person or people you'd like to remove from owner role.
6. Above the list of members, use the drop-down menu and click Change role.
7. Select **Member**, then click **Change role**.

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 each repository in use, verify that Multi-Factor Authentication is enforced for contributors and is the only way to authenticate, by doing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.
4. Under "Authentication", check if **Require two-factor authentication for everyone in your organization** is checked. If so, you are compliant.

Remediation:

For each repository in use, enforce Multi-Factor Authentication is the only way to authenticate for contributors, by doing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.

4. Under "Authentication", select **Require two-factor authentication for everyone in your organization**, then click **Save**.

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.			

1.3.5 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 GitHub platform, verify that Multi-Factor Authentication is enforced and is the only way to authenticate, by doing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.
4. Under "Authentication", check if **Require two-factor authentication for everyone in your organization** is checked. If so, you are compliant.

Remediation:

For every organization that exists in your GitHub platform, enforce Multi-Factor Authentication and define it as the only way to authenticate, by doing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.

2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.
4. Under "Authentication", select **Require two-factor authentication for everyone in your organization**, then click **Save**.
5. If prompted, read the information about members and outside collaborators who will be removed from the organization. Type your organization's name to confirm the change, then click **Remove members & require two-factor authentication**.

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

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Click the name of your organization and under your organization name, click **People**.
3. On the People tab, click **Invitations**. Verify that each invitation email is company approved by your company.

Remediation:

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

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Click the name of your organization and under your organization name, click **People**.

3. On the People tab, click **Invitations**. Next to the username or email address of the person whose invitation you'd like to cancel, click **Edit invitation**.
4. To cancel the user's invitation to join your organization, click **Cancel invitation**.

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 repository in use, verify there are two administrators by performing the following:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Access" section of the sidebar, click **Collaborators & teams**.
4. Under **Manage access**, verify that there are only 2 members with Admin permission.

Remediation:

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

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Access" section of the sidebar, click **Collaborators & teams**.
4. Under **Manage access**, find the team or person whose you'd like to revoke admin permissions, then select the Role drop-down and click a new role.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.7 <u>Centralize Access Control</u> Centralize access control for all enterprise assets through a directory service or SSO provider, where supported.			
v7	14 <u>Controlled Access Based on the Need to Know</u> Controlled Access Based on the Need to Know			

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 repositories by doing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Access" section of the sidebar, click **Member privileges**.
4. Under "Base permissions", verify that it is set to "Read" or "None". If it does, you are compliant.

Remediation:

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

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.

3. In the "Access" section of the sidebar, click **Member privileges**.
4. Under "Base permissions", use the drop-down to select new base permissions - "Read" or "None".
5. Review the changes. To confirm, click **Change default permission to PERMISSION**.

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:

Only if you have an enterprise account, view the enterprise organization profile page and ensure it has a "Verified" badge in it.

Remediation:

Only if you have an enterprise account, verify the organization's domains and secure a "Verified" badge next to its name by performing the following:

1. In the top-right corner of GitHub.com, click your profile photo, then click **Your enterprises**.
2. In the list of enterprises, click the enterprise you want to view. Then in the enterprise account sidebar, click **Settings**.
3. Under "Settings", click **Verified & approved domains**.
4. Click **Add a domain**.
5. In the domain field, type the domain you'd like to verify, then click **Add domain**.
6. Follow the instructions under **Add a DNS TXT record** to create a DNS TXT record with your domain hosting service. Wait for your DNS configuration to change, which may take up to 72 hours. You can confirm your DNS configuration has changed by running the `dig` command on the command line, replacing ENTERPRISE-ACCOUNT with the name of your enterprise account, and example.com with the domain you'd like to verify. You should see your new TXT record listed in the command output.

```
dig _github-challenge-ENTERPRISE-ACCOUNT.DOMAIN-NAME +nostats +nocomments  
+nocmd TXT
```

7. After confirming your TXT record is added to your DNS, follow steps one through three above to navigate to your enterprise account's approved and verified domains.
8. To the right of the domain that's pending verification, click the 3-dots, then click **Continue verifying**. Click **Verify**.
9. Optionally, after the "Verified" badge is visible on your organizations' profiles, delete the TXT entry from the DNS record at your domain hosting service.

References:

1. <https://docs.github.com/en/organizations/managing-organization-settings/verifying-or-approving-a-domain-for-your-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.			
v8	16.11 <u>Leverage Vetted Modules or Services for Application Security Components</u> Leverage vetted modules or services for application security components, such as identity management, encryption, and auditing and logging. Using platform features in critical security functions will reduce developers' workload and minimize the likelihood of design or implementation errors. Modern operating systems provide effective mechanisms for identification, authentication, and authorization and make those mechanisms available to applications. Use only standardized, currently accepted, and extensively reviewed encryption algorithms. Operating systems also provide mechanisms to create and maintain secure audit logs.			
v7	18.1 <u>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:

Only if you have an enterprise account, ensure Source Code Management email notifications are restricted to approved domains only by performing the following:

1. In the top-right corner of GitHub.com, click your profile photo, then click **Your enterprises**.
2. In the list of enterprises, click the enterprise you want to view. Then in the enterprise account sidebar, click **Settings**.
3. Under "Settings", click **Verified & approved domains**.
4. Under "Notification preferences", verify that **Restrict email notifications to only approved or verified domains** is selected.

Remediation:

Only if you have an enterprise account, restrict Source Code Management email notifications to approved domains only by performing the following:

1. In the top-right corner of GitHub.com, click your profile photo, then click **Your enterprises**.
2. In the list of enterprises, click the enterprise you want to view. Then in the enterprise account sidebar, click **Settings**.
3. Under "Settings", click **Verified & approved domains**.
4. Under "Notification preferences", select **Restrict email notifications to only approved or verified domains**.

5. Click **Save**.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	13.11 <u>Tune Security Event Alerting Thresholds</u> Tune security event alerting thresholds monthly, or more frequently.			

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:

Only if you have an enterprise account, verify that the enterprise organization has an SSH Certificate Authority server and provides an SSH certificate with which to sign keys by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.
4. Verify that there's an SSH certificate authority listed there.

Remediation:

Only if you have an enterprise account, deploy an SSH Certificate Authority server and configure it to provide an SSH certificate with which to sign keys by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.
4. To the right of "SSH Certificate Authorities", click **New CA**.
5. Under "Key," paste your public SSH key.
6. Click **Add CA**.
7. Click **Save**.

CIS Controls:

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

Only if you have an enterprise account, in every organization of yours, ensure access is allowed only by IP allowlist, and that access is forbidden for all other IPs by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.
4. Verify that there's an IP address, or a range of addresses in CIDR notation listed. Also verify that **Enable IP allow list** is selected.

Remediation:

Only if you have an enterprise account, create an IP allowlist and forbid all other IPs from accessing the source code by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Security" section of the sidebar, click **Authentication security**.

4. At the bottom of the "IP allow list" section, enter an IP address, or a range of addresses in CIDR notation. Optionally, enter a description of the allowed IP address or range.
5. Click **Add**.
6. After that, under "IP allow list", select **Enable IP allow list**.
7. Click **Save**.

References:

1. <https://docs.github.com/en/organizations/keeping-your-organization-secure/managing-allowed-ip-addresses-for-your-organization>

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 repository in use, ensure code anomalies relevant to the organization are promptly investigated.

Remediation:

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

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.		●	●

1.4 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 or Github 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:

- a. For GitHub Apps, you are compliant by default. That is because by default only organization owners and administrators can install them.
- b. For OAuth Apps, perform the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the "Third-party Access" section of the sidebar, click **OAuth application policy**.
4. Under "Third-party application access policy" verify that the policy status is **Access restricted**.

Remediation:

Require an administrator approval for every installed application:

- a. For GitHub Apps, you are compliant by default. That is because by default only organization owners and administrators can install them.
- b. For OAuth Apps, perform the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.

2. Next to the organization, click **Settings**.
3. In the "Third-party Access" section of the sidebar, click **OAuth application policy**.
4. Under "Third-party application access policy", click **Setup application access restrictions**.
5. After you review the information about third-party access restrictions, click **Restrict third-party application access**.

Default Value:

Maintainers are organization owners.

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.			
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.			

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.

Remediation:

Review all stale applications and periodically remove them.

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 application has the least privilege needed:

- a. For GitHub Apps, perform the following:
 1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
 2. Next to the organization, click **Settings**.
 3. In the "Integrations" section of the sidebar, click **GitHub Apps**.
 4. Next to every GitHub App, click **Configure**.
 5. Review the GitHub App's permissions and repository access. Verify that the App permissions are the least possible and that it can access only necessary repositories.

Remediation:

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

- a. For GitHub Apps, perform the following:
 1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
 2. Next to the organization, click **Settings**.
 3. In the "Integrations" section of the sidebar, click **GitHub Apps**.
 4. Next to every GitHub App, click **Configure**.

5. Review the GitHub App's permissions and repository access. Edit the permissions granted to the least possible. For example, restrict the number of repositories the App can access.
6. Click **Save**.

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 used(over HTTPS):

1. Navigate to your organization or repository and select **Settings**.
2. Select **Webhooks** on the side menu.
3. Ensure all webhooks starts with 'https'.

Remediation:

Perform the following to secure all webhooks used(over HTTPS):

1. Navigate to your organization or repository and select **Settings**.
2. Select **Webhooks** on the side menu.
3. Find the webhooks that starts with 'http' and not 'https'.
4. Ensure the endpoint (URL) of the webhook listens to secured port (443) and uses certificate.
5. Click **Edit**.
6. Change the payload URL to https and ensure **Enable SSL verification** is checked.

7. Click **Update webhook**.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
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	2.9 <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.			●

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:

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 (enterprise only):

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Security" section of the sidebar, click **Code security and analysis**.
4. Scroll down to the bottom of the page. If you see a **Disable** button, it means that secret scanning is already enabled for the repository.

Remediation:

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

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the "Security" section of the sidebar, click **Code security and analysis**.
4. Scroll down to the bottom of the page and click **Enable** for secret scanning.

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	<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.			

1.5.2 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:

Verify that a CI instructions scanning tool is set to identify and prevent misconfigurations and insecure instructions and that it scans all CI pipelines.

Remediation:

Set 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:

For every repository that holds IaC instructions files, verify that a scanning tool is set to identify and prevent misconfigurations and insecure instructions.

Remediation:

For every repository that holds IaC instructions files, set a scanning tool 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>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.			

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.		●	●

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:

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 GitHub.com, navigate to the main page of the repository.
2. Under the repository name, click **Security**.
3. Verify that "Code scanning alerts" is **Enabled** or that there is a workflow that scans your 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 GitHub.com, navigate to the main page of the repository.
2. Under the repository name, click **Security**.
3. To the right of "Code scanning alerts", click **Set up code scanning**.
4. Under "Get started with code scanning", click Set up this workflow on a workflow of your choice.
5. To customize how code scanning scans your code, edit the workflow.
6. Use the **Start commit** drop-down and type a commit message.
7. Choose whether you'd like to commit directly to the default branch or create a new branch and start a pull request.
8. Click **Commit new file** or **Propose new file**.

Additional Information:

All public repositories in all plans can use code scanning in GitHub. In private repositories, only GitHub Enterprise can use code scanning.

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.			●
v8	16.13 <u>Conduct Application Penetration Testing</u> Conduct application penetration testing. For critical applications, authenticated penetration testing is better suited to finding business logic vulnerabilities than code scanning and automated security testing. Penetration testing relies on the skill of the tester to manually manipulate an application as an authenticated and unauthenticated user.			●
v7	20.6 <u>Use Vulnerability Scanning and Penetration Testing Tools in Concert</u> Use vulnerability scanning and penetration testing tools in concert. The results of vulnerability scanning assessments should be used as a starting point to guide and focus penetration testing efforts.		●	●

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:

Verify that scanners are set to monitor, identify, and prevent open-source vulnerabilities in packages by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**. In the "Security" section of the sidebar, click **Code security and analysis**.
3. Verify that the Dependabot alerts is enabled and that **Enable by default for new repositories** is checked.

It is also recommended to verify that you're using another additional solution for package scanning because GitHub doesn't always recognize the vulnerabilities.

Remediation:

Set scanners that will monitor, identify, and prevent open-source vulnerabilities in packages by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**. In the "Security" section of the sidebar, click **Code security and analysis**.
3. Under "Code security and analysis", to the right of Dependabot alerts, click **Disable all** or **Enable all**.

4. Check the **Enable by default for new repositories** option and then click **Enable Dependabot alerts**.

It is also recommended to use another additional solution for package scanning because GitHub doesn't always recognize the vulnerabilities.

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.			

1.5.6 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.

Remediation:

Designate a license scanning tool to identify open-source license problems and fix them and scan every package you use.

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.			
v8	7.6 <u>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.			

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.			

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.github.com/en/actions/using-jobs/assigning-permissions-to-jobs>

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.			

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.10 Apply Secure Design Principles in Application Architectures</u> 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	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.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	5.1 Establish Secure Configurations Maintain documented, standard security configuration standards for all authorized operating systems and software.			
v7	5.2 Maintain Secure Images Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.			

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.

Remediation:

Keep logs of the build environment. For example, use the .buildinfo file for Debian build workers. Also, store the logs in a centralized organizational log store.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.14 Log Sensitive Data Access Log sensitive data access, including modification and disposal.			●
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).			●

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.

Remediation:

Automate the deployment of the build environment.

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.

Remediation:

Restrict access to the build environment to trusted and qualified users.

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.	●	●	●

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.

Remediation:

Require authentication to access the build environment and disable anonymous access.

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, for example in github it could be on an organization level or a repository level and there is also control on whether these secrets are passed to forked pull request. 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:

For each build tool in use, review the secrets defined and the permission scopes they are assigned.

Remediation:

For each build tool, review the secrets defined and their permissions scope and change over permissive scopes to more restrictive ones based on the required access.

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.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	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.			

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 automatically scanned for vulnerabilities.

Remediation:

Set an automated vulnerability scanning for your build infrastructure.

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.			
v8	7.6 <u>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	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.			

2.1.9 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:

For each build tool, ensure the password used is not the default one.

Remediation:

For each build tool, change the default password.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	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.			
v7	4.2 Change Default Passwords 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).

Remediation:

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

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.

Remediation:

Set the minimum number of administrators in the build environment.

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.			

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.

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.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.12 <u>Segment Data Processing and Storage Based on Sensitivity</u> 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.			
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.			

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.

Remediation:

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

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.			

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:

For each build worker, ensure it has the least responsibility possible, preferably only one duty.

Remediation:

For each build worker, limit its responsibility to one duty.

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.			

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.

Remediation:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	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 anti-malware software installed, configuration compliance with the enterprise's secure configuration process, and ensuring the operating system and applications are up-to-date.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	12.12 <u>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	3.14 <u>Log Sensitive Data Access</u> Log sensitive data access, including modification and disposal.			●

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:

For each build worker, ensure the environmental sources it uses are scanned for vulnerabilities.

Remediation:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	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.			
v7	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.			

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 Github.

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.

Remediation:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.1 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	<u>5.5 Implement Automated Configuration Monitoring Systems</u> Utilize a Security Content Automation Protocol (SCAP) compliant configuration monitoring system to verify all security configuration elements, catalog approved exceptions, and alert when unauthorized changes occur.			

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.

Remediation:

Set resources consumption monitoring for each build worker.

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>5.5 Implement Automated Configuration Monitoring Systems</u> Utilize a Security Content Automation Protocol (SCAP) compliant configuration monitoring system to verify all security configuration elements, catalog approved exceptions, and alert when unauthorized changes occur.			

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.

Remediation:

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

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.			
v7	5.4 Deploy System Configuration Management Tools Deploy system configuration management tools that will automatically enforce and redeploy configuration settings to systems at regularly scheduled intervals.			

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.

Remediation:

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

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.			

2.3.3 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.

Remediation:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>3.12 Segment Data Processing and Storage Based on Sensitivity</u> 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.

Remediation:

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

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.			●

2.3.5 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.

Remediation:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.7 Allowlist Authorized Scripts 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	4.7 Limit Access to Script Tools Limit access to scripting tools (such as Microsoft PowerShell and Python) to only administrative or development users with the need to access those capabilities.		●	●

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	<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>3.1 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.			

2.3.7 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.

Remediation:

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

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
- Level 2

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	18.4 <u>Validate Security Measures</u> Validate security measures after each penetration test. If deemed necessary, modify rulesets and capabilities to detect the techniques used during testing.			●
v7	20.4 <u>Include Tests for Presence of Unprotected System Information and Artifacts</u> Include tests for the presence of unprotected system information and artifacts that would be useful to attackers, including network diagrams, configuration files, older penetration test reports, e-mails or documents containing passwords or other information critical to system operation.		●	●

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.

Remediation:

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

References:

1. <https://argon.io/blog/pipeline-composition-analysis-how-your-ci-pipeline-presents-new-opportunities-for-attackers/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	7.6 Perform Automated Vulnerability Scans of Externally-Exposed 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.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	20.2 <u>Conduct Regular External and Internal Penetration Tests</u> Conduct regular external and internal penetration tests to identify vulnerabilities and attack vectors that can be used to exploit enterprise systems successfully.		●	●

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.

Remediation:

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

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.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	20.4 <u>Include Tests for Presence of Unprotected System Information and Artifacts</u> Include tests for the presence of unprotected system information and artifacts that would be useful to attackers, including network diagrams, configuration files, older penetration test reports, e-mails or documents containing passwords or other information critical to system operation.		●	●

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.

Remediation:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	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.			
v7	3.6 Compare Back-to-back Vulnerability Scans Regularly compare the results from back-to-back vulnerability scans to verify that vulnerabilities have been remediated in a timely manner.			

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.

Remediation:

For each pipeline, configure it to produce a Software Bill of Materials on every run.

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	2 <u>Inventory and Control of Software Assets</u> Inventory and Control of Software Assets			

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	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.			

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:

For every GitHub action (building block of a GitHub workflow), ensure verification before use by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.
2. Next to the organization, click **Settings**.
3. In the left sidebar, click **Actions**, then click **General**.
4. Under "Policies", ensure **Allow OWNER, and select non-OWNER, actions and reusable workflows**, and **Allow actions created by GitHub** are checked.

Alternatively, perform the following for each repository where GitHub actions are used:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the left sidebar, click **Actions**, then click **General**.
4. Under "Actions permissions", ensure **Allow OWNER, and select non-OWNER, actions and reusable workflows** and **Allow actions created by GitHub** are checked.

Remediation:

Verify every GitHub action (building block of a GitHub workflow) in use by performing the following:

1. In the top right corner of GitHub.com, click your profile photo, then click **Your organizations**.

2. Next to the organization, click **Settings**.
3. In the left sidebar, click **Actions**, then click **General**.
4. Under "Policies", check **Allow OWNER**, and select **non-OWNER, actions and reusable workflows**, and then **Allow actions created by GitHub**.
5. Click **Save**.

Alternatively, perform the following for each repository where GitHub actions are used:

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**.
3. In the left sidebar, click **Actions**, then click **General**.
4. Under "Actions permissions", check **Allow OWNER**, and select **non-OWNER, actions and reusable workflows**, and then **Allow actions created by GitHub**.
5. Click **Save**.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
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	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.			

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	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	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.			

Controls Version	Control	IG 1	IG 2	IG 3
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.		●	●

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	2.3 Address Unauthorized Software Ensure that unauthorized software is either removed from use on enterprise assets or receives a documented exception. Review monthly, or more frequently.			
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	2.3 Utilize Software Inventory Tools Utilize software inventory tools throughout the organization to automate the documentation of all software on business systems.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	2.6 Address unapproved software Ensure that unauthorized software is either removed or the inventory is updated in a timely manner			

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 open-source component, ensure its dependencies are monitored.

Remediation:

For each open-source component, monitor its dependencies.

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	2.3 Utilize Software Inventory Tools Utilize software inventory tools throughout the organization to automate the documentation of all software on business systems.			
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.			

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.

Remediation:

For each package to be downloaded, configure it to be downloaded from a trusted source.

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.			
v8	2.7 Allowlist Authorized Scripts 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.			

Controls Version	Control	IG 1	IG 2	IG 3
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.			●
v7	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.			●

3.1.6 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.

Remediation:

For every artifact supplied, require and verify a signed Software Bill of Materials from its supplier.

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.			

Controls Version	Control	IG 1	IG 2	IG 3
v8	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.			
v7	2.1 Maintain Inventory of Authorized Software 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	2.2 Ensure Software is Supported by Vendor 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	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 bi-annually, or more frequently.			
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.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	2.1 <u>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	2.3 <u>Utilize Software Inventory Tools</u> Utilize software inventory tools throughout the organization to automate the documentation of all software on business systems.			

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	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.			

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.7 Allowlist Authorized Scripts 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	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	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.			●
v7	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.			●

3.2 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.			

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:

Ensure automatic scanning of packages for vulnerabilities is enabled.

Remediation:

Set automatic scanning of packages for vulnerabilities.

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.

Remediation:

Set automatic package scanning for license implications.

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.			

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.		●	●

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	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.			

Controls Version	Control	IG 1	IG 2	IG 3
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.			

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.

Steps from GitHub Documentation:

You can follow the steps below to sign artifacts in GitHub actions. The trick involves loading in your private key into GitHub Actions using the gpg command-line commands. Export your gpg private key from the system that you have created it.

Find your key-id (using `gpg --list-secret-keys --keyid-format=long`)

Export the gpg secret key to an ASCII file using `gpg --export-secret-keys -a > secret.txt`

Edit `secret.txt` using a plain text editor, and replace all newlines with a literal `"\n"` until everything is on a single line

Set up GitHub Actions secrets

Create a secret called `OSSRH_GPG_SECRET_KEY` using the text from your edited `secret.txt` file (the whole text should be in a single line)

Create a secret called `OSSRH_GPG_SECRET_KEY_PASSWORD` containing the password for your gpg secret key

Create a GitHub Actions step to install the gpg secret key

Add an action similar to:

```
- id: install-secret-key
  name: Install gpg secret key
  run: |
    cat <(echo -e "${{ secrets.OSSRH_GPG_SECRET_KEY }}") | gpg --batch --import
    gpg --list-secret-keys --keyid-format LONG
```

Verify that the secret key is shown in the GitHub Actions logs

You can remove the output from list secret keys if you are confident that this action will work, but it is better to leave it in there

Bring it all together, and create a GitHub Actions step to publish

Add an action similar to:

```
- id: publish-to-central
  name: Publish to Central Repository
  env:
    MAVEN_USERNAME: ${ secrets.OSSRH_USERNAME }
    MAVEN_PASSWORD: ${ secrets.OSSRH_TOKEN }
  run: |
    mvn \
      --no-transfer-progress \
      --batch-mode \
      -Dgpg.passphrase=${ secrets.OSSRH_GPG_SECRET_KEY_PASSWORD } \
      clean deploy
```

After a couple of hours, verify that the artifact got published to The Central Repository

Default Value:

Artifacts are not signed by Default.

References:

1. <https://docs.github.com/en/code-security/supply-chain-security/end-to-end-supply-chain/securing-builds>
2. <https://gist.github.com/sualeh/ae78dc16123899d7942bc38baba5203c>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	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.		●	●

Controls Version	Control	IG 1	IG 2	IG 3
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.		●	●

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	<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.		●	●

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 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.			

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.7 Allowlist Authorized Scripts 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	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.			

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.

Remediation:

Allow only trusted and qualified users to upload new artifacts and limit them in number.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.7 Centralize Access Control Centralize access control for all enterprise assets through a directory service or SSO provider, where supported.			
v7	4.7 Limit Access to Script Tools Limit access to scripting tools (such as Microsoft PowerShell and Python) to only administrative or development users with the need to access those capabilities.			

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.

Remediation:

For each package registry in use, enforce Multi-Factor Authentication as the only way to authenticate.

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.

4.2.5 Ensure anonymous access to artifacts is revoked (Manual)

Profile Applicability:

- Level 1

Description:

For GitHub Private or Internal repositories anonymous access is not available. Verify that all repos 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:

To Audit the existing settings:

1. On your GitHub Enterprise Server instance, navigate to the main page of the repository.
2. Under your repository name, click Settings
3. Under "Danger Zone", to the right of to "Change repository visibility", click Change visibility.

Review the current selection.

Remediation:

Changing a repository's visibility

1. On your GitHub Enterprise Server instance, navigate to the main page of the repository.
2. Under your repository name, click Settings
3. Under "Danger Zone", to the right of to "Change repository visibility", click Change visibility.
4. Select a visibility.
5. Choose
 - Mark private
 - Make Internal

6. To verify that you're changing the correct repository's visibility, type the name of the repository you want to change the visibility of.

References:

1. <https://docs.github.com/en/enterprise-server@3.3/repositories/managing-your-repositorys-settings-and-features/managing-repository-settings/setting-repository-visibility>

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	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.			

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 repository that you administer on GitHub, you can see an overview of every team or person with access to the repository. From the overview, you can also invite new teams or people, change each team or person's role for the repository, or remove access to the repository.

Remediation:

Set the minimum number of administrators in your package registry.

To accomplish this:

For each repository that you administer on GitHub, you can see an overview of every team or person with access to the repository. From the overview, choose Manage access and provide access to the appropriate people or teams.

References:

1. <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/managing-repository-settings/managing-teams-and-people-with-access-to-your-repository>

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>4.1 Maintain Inventory of Administrative Accounts</u> Use automated tools to inventory all administrative accounts, including domain and local accounts, to ensure that only authorized individuals have elevated privileges.			

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 GitHub, navigate to a pull request
2. On the pull request, click <> Commits and view the detailed information regarding the signature.

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.github.com/en/authentication/managing-commit-signature-verification/about-commit-signature-verification>
2. <https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-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.

On GitHub, navigate to a pull request

On the pull request, click <> Commits and view the detailed information regarding the signature.

Remediation:

For each artifact, sign and validate each version before uploading or using the artifact.

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.			

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	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.			

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.
Search the audit log with
repo category actions

Remediation:

Audit the changes to the package registry configuration.

Default Value:

GitHub audits this by default.

References:

1. <https://docs.github.com/en/organizations/keeping-your-organization-secure/managing-security-settings-for-your-organization/reviewing-the-audit-log-for-your-organization>

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.			

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).			
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	1.1 <u>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.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	1.5 Maintain Asset Inventory Information 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.			

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	6.2 Activate audit logging Ensure that local logging has been enabled on all systems and networking devices.	●	●	●

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	6.7 Centralize Access Control Centralize access control for all enterprise assets through a directory service or SSO provider, where supported.		●	●
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.			●

Controls Version	Control	IG 1	IG 2	IG 3
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.			

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.			
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.			

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>16.7 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.			

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	16.7 <u>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	3.6 <u>Compare Back-to-back Vulnerability Scans</u> Regularly compare the results from back-to-back vulnerability scans to verify that vulnerabilities have been remediated in a timely manner.			

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:

For each deployment tool, ensure the password is not the default one.

Remediation:

For each deployment tool, change the password.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	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.			
v7	4.2 Change Default Passwords Before deploying any new asset, change all default passwords to have values consistent with administrative level accounts.			
v7	4.4 Use Unique Passwords 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 team 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)	☐	☐
2	Build Pipelines		
2.1	Build Environment		

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
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)	☐	☐
2.2.3	Ensure the duties of each build worker are segregated (Manual)	☐	☐
2.2.4	Ensure build workers have minimal network connectivity (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
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)	☐	☐
2.4	Pipeline Integrity		
2.4.1	Ensure all artifacts on all releases are signed (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
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)	☐	☐
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)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
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)	☐	☐
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)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
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)	☐	☐
5.1.5	Scan Infrastructure as Code (IaC) (Manual)	☐	☐
5.1.6	Ensure deployment configuration manifests are verified (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
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
1.1.6	Ensure code owners are set for extra sensitive code or configuration	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐
1.1.19	Ensure any changes to branch protection rules are audited	☐	☐
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 team creation is limited to specific members	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization	☐	☐
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	☐	☐
2.1.2	Ensure all aspects of the pipeline infrastructure and configuration are immutable	☐	☐
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	☐	☐
3.1.3	Ensure signed metadata of the build process is required and verified	☐	☐
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
4.2.5	Ensure anonymous access to artifacts is revoked	☐	☐
5.1.2	Ensure changes in deployment configuration are audited	☐	☐
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	☐	☐
1.1.6	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	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐
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	☐	☐
1.2.1	Ensure all public repositories contain a SECURITY.md file	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for	☐	☐
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 team creation is limited to specific members	☐	☐

Recommendation		Set Correctly	
		Yes	No
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)	☐	☐
1.3.8	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	☐	☐
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	☐	☐
1.5.6	Ensure scanners are in place for open-source license issues in used packages	☐	☐
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	☐	☐
2.1.7	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	☐	☐

Recommendation		Set Correctly	
		Yes	No
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	☐	☐
2.2.8	Ensure resource consumption of build workers is monitored	☐	☐
2.3.1	Ensure all build steps are defined as code	☐	☐
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	☐	☐
2.3.7	Ensure pipelines are automatically scanned for vulnerabilities	☐	☐
2.4.1	Ensure all artifacts on all releases are signed	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked	☐	☐
2.4.3	Ensure dependencies are validated before being used	☐	☐
2.4.4	Ensure the build pipeline creates reproducible artifacts	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers	☐	☐
3.1.3	Ensure signed metadata of the build process is required and verified	☐	☐
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	☐	☐
3.2.2	Ensure packages are automatically scanned for known vulnerabilities	☐	☐
3.2.3	Ensure packages are automatically scanned for license implications	☐	☐

Recommendation		Set Correctly	
		Yes	No
3.2.4	Ensure packages are automatically scanned for ownership change	☐	☐
4.1.1	Ensure all artifacts are signed by the build pipeline itself	☐	☐
4.1.2	Ensure artifacts are encrypted before distribution	☐	☐
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)	☐	☐
4.2.5	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	☐	☐
4.3.4	Ensure webhooks of the repository are secured	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
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	☐	☐
5.2.2	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	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐
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	☐	☐
1.2.1	Ensure all public repositories contain a SECURITY.md file	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for	☐	☐
1.2.7	Ensure inactive repositories are reviewed and archived periodically	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.3.1	Ensure inactive users are reviewed and removed periodically	☐	☐
1.3.2	Ensure team 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	☐	☐
1.3.8	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	☐	☐
1.3.13	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	☐	☐
1.4.4	Ensure only secured webhooks are used	☐	☐
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	☐	☐
1.5.6	Ensure scanners are in place for open-source license issues in used packages	☐	☐

Recommendation		Set Correctly	
		Yes	No
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	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐
2.2.1	Ensure build workers are single-used	☐	☐
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	☐	☐
2.2.8	Ensure resource consumption of build workers is monitored	☐	☐
2.3.1	Ensure all build steps are defined as code	☐	☐
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	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.4.1	Ensure all artifacts on all releases are signed	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked	☐	☐
2.4.3	Ensure dependencies are validated before being used	☐	☐
2.4.4	Ensure the build pipeline creates reproducible artifacts	☐	☐
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	☐	☐
3.1.3	Ensure signed metadata of the build process is required and verified	☐	☐
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	☐	☐
3.2.2	Ensure packages are automatically scanned for known vulnerabilities	☐	☐
3.2.3	Ensure packages are automatically scanned for license implications	☐	☐
3.2.4	Ensure packages are automatically scanned for ownership change	☐	☐
4.1.1	Ensure all artifacts are signed by the build pipeline itself	☐	☐
4.1.2	Ensure artifacts are encrypted before distribution	☐	☐
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)	☐	☐
4.2.5	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	☐	☐
4.3.4	Ensure webhooks of the repository are secured	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
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	☐	☐
5.2.2	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 Unmapped Recommendations

Recommendation		Set Correctly	
		Yes	No
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	☐	☐
1.1.20	Ensure branch protection is enforced on the default branch	☐	☐
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	☐	☐
1.2.6	Ensure all code projects are tracked for changes in visibility status	☐	☐
1.3.10	Ensure Source Code Management (SCM) email notifications are restricted to verified domains	☐	☐
2.1.1	Ensure each pipeline has a single responsibility	☐	☐
2.1.10	Ensure webhooks of the build environment are secured	☐	☐
2.2.2	Ensure build worker environments and commands are passed and not pulled	☐	☐
2.2.3	Ensure the duties of each build worker are segregated	☐	☐
2.2.5	Ensure run-time security is enforced for build workers	☐	☐
2.3.3	Ensure output is written to a separate, secured storage repository	☐	☐
2.4.6	Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced	☐	☐

Recommendation		Set Correctly	
		Yes	No
3.2.1	Ensure an organization-wide dependency usage policy is enforced	☐	☐
4.1.3	Ensure only authorized platforms have decryption capabilities of artifacts	☐	☐
4.2.1	Ensure the authority to certify artifacts is limited	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐
4.3.3	Ensure changes in package registry configuration are audited	☐	☐
5.1.1	Ensure deployment configuration files are separated from source code	☐	☐
5.2.1	Ensure deployments are automated	☐	☐

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 team 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	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed	☐	☐
2.1.1	Ensure each pipeline has a single responsibility	☐	☐
2.1.7	Ensure build secrets are limited to the minimal necessary scope	☐	☐
2.1.9	Ensure default passwords are not used	☐	☐

Recommendation		Set Correctly	
		Yes	No
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	☐	☐
2.3.1	Ensure all build steps are defined as code	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers	☐	☐
3.1.3	Ensure signed metadata of the build process is required and verified	☐	☐
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	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
5.1.6	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	☐	☐
1.1.4	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	☐	☐
1.1.15	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	☐	☐
1.1.18	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	☐	☐

Recommendation		Set Correctly	
		Yes	No
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 team 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	☐	☐
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	☐	☐
1.5.5	Ensure scanners are in place for open-source vulnerabilities in used packages	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.5.6	Ensure scanners are in place for open-source license issues 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	☐	☐
2.1.6	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	☐	☐
2.2.1	Ensure build workers are single-used	☐	☐
2.2.2	Ensure build worker environments and commands are passed and not pulled	☐	☐
2.2.3	Ensure the duties of each build worker are segregated	☐	☐
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	☐	☐
2.2.8	Ensure resource consumption of build workers is monitored	☐	☐
2.3.1	Ensure all build steps are defined as code	☐	☐
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	☐	☐
2.3.6	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	☐	☐
2.4.4	Ensure the build pipeline creates reproducible artifacts	☐	☐
2.4.5	Ensure pipeline steps produce a Software Bill of Materials (SBOM)	☐	☐
2.4.6	Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced	☐	☐
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	☐	☐
3.1.3	Ensure signed metadata of the build process is required and verified	☐	☐
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	☐	☐
3.2.3	Ensure packages are automatically scanned for license implications	☐	☐
3.2.4	Ensure packages are automatically scanned for ownership change	☐	☐
4.1.1	Ensure all artifacts are signed by the build pipeline itself	☐	☐
4.1.2	Ensure artifacts are encrypted before distribution	☐	☐

Recommendation		Set Correctly	
		Yes	No
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)	☐	☐
4.2.5	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	☐	☐
5.1.4	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	☐	☐
1.1.15	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	☐	☐
1.1.18	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	☐	☐

Recommendation		Set Correctly	
		Yes	No
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 team 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	☐	☐
1.3.10	Ensure Source Code Management (SCM) email notifications are restricted to verified domains	☐	☐
1.3.11	Ensure an organization provides SSH certificates	☐	☐
1.3.12	Ensure Git access is limited based on IP addresses	☐	☐
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	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.4.4	Ensure only secured webhooks are used	☐	☐
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	☐	☐
1.5.6	Ensure scanners are in place for open-source license issues 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	☐	☐
2.1.3	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	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐
2.2.1	Ensure build workers are single-used	☐	☐
2.2.2	Ensure build worker environments and commands are passed and not pulled	☐	☐
2.2.3	Ensure the duties of each build worker are segregated	☐	☐
2.2.4	Ensure build workers have minimal network connectivity	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.2.5	Ensure run-time security is enforced for build workers	☐	☐
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	☐	☐
2.2.8	Ensure resource consumption of build workers is monitored	☐	☐
2.3.1	Ensure all build steps are defined as code	☐	☐
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	☐	☐
2.4.1	Ensure all artifacts on all releases are signed	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked	☐	☐
2.4.3	Ensure dependencies are validated before being used	☐	☐
2.4.4	Ensure the build pipeline creates reproducible artifacts	☐	☐
2.4.5	Ensure pipeline steps produce a Software Bill of Materials (SBOM)	☐	☐
2.4.6	Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced	☐	☐
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	☐	☐

Recommendation		Set Correctly	
		Yes	No
3.1.3	Ensure signed metadata of the build process is required and verified	☐	☐
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	☐	☐
3.2.3	Ensure packages are automatically scanned for license implications	☐	☐
3.2.4	Ensure packages are automatically scanned for ownership change	☐	☐
4.1.1	Ensure all artifacts are signed by the build pipeline itself	☐	☐
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)	☐	☐
4.2.5	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	☐	☐

Recommendation		Set Correctly	
		Yes	No
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 Unmapped Recommendations

Recommendation		Set Correctly	
		Yes	No
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	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐

Appendix: Change History

Date	Version	Changes for this version