Component		Method	Procedure (example)	Attacker's Prerequisites	Impact	Prevention (injection prevention)	Prevention (trigger prevention)	Prevention (impact reduction)	Detection	Treceability
Source Repository	**A: Commits**	An attacker commits malcious changes using compromised accounts to *feature* branches	An attacker compromises developer's device. Then they commit malicious changes to a feature branch using an authorized SSH key. The changes might be merged to the main branch after reviews, or actions are performed based on modified actions workflow file.	- Valid write access to source repo	- Changes to source code - Changes to GitHub Actions workflow file	- Force 2FA - Force 2 person review(branch protection, CODEOWNERS) - Sign all commits - Network restriction to source repo - Continuous scan for inactive users and remove them			- Sign all commits - Audit log	- Sign all commits - Audit Log
		An attacker commits malicious changes using stolen accounts to *main* branch	An attacker compromises admin's device, who has priviledge access to main branch. They commit malicious changes to the main branch without review using a priviledged SSH key.	- Compromise of admin accounts	- Changes to source code w/o review - Changes to GitHub Actions workflow file w/o review	- Force 2FA - Set appropriate branch protection rule and apply it to all contributers including admins - Limit number of admins - Sign all commits - Network restriction to source repo - Continuous scan for inactive users and remove them			- Sign all commits - Audit Log	- Sign all commits - Prohibit force push - Audit Log
		An attacker searches old, inactive, or less-secure branches/repositories for seinsitive data or possible vulnerabilities	If old branches with sensitive data and less security remains in source repository, attacker can search old branches for sensitive data and possible vulnerabilities.	- Valid read access to source repo	- Information leakage	- Continuous scan/tracking of branches to remove old ones - Network restriction to source repo			- Sign all commits - Audit Log	- Sign all commits - Audit Log
			If old branches with lower security level remain in source repository (eg: not requiring 2 person review) and are still deployed to prod environment, An attacker can commits to them and deploy to production environment.	- Valid write access to source repo	- RCE	- Continuous scan/tracking of branches to remove old ones - Network restriction to source repo			- Sign all commits - Audit Log	- Sign all commits - Audit Log
	B: Review	Self-approval of malicious commits by attacker	An attacker commits changes to feature branch. Then they approve the changes by themselve.	- Valid write access to source repo (feature branch) - Valid approval permission - Settings which allows self-approval	- Changes to source code w/o review - Changes to GitHub Actions workflow file w/o review	- Network restriction to source repo - Force 2 person review(branch protection, CODEOWNERS)			- Sign all commits - Audit Log	- Sign all commits - Audit Log
		Approval by compromised accounts	An attacker commits changes to feature branch. Then they approves the changes using compromised other accounts.	- Valid write access to source repo (feature branch) - Compromise of valid approver's account	- Changes to source code w/o review - Changes to GitHub Actions workflow file w/o review	- Force 2FA - Network Restriction - Force 2 person review(branch protection, CODEOWNERS)			- Sign all commits - Audit Log	- Sign all commits - Audit Log
		Add malicious changes after valid approval	An attacker commits valid changes to feature branch. Valid approvers approve the commits. Then attacker additionally commits changes to approved PR, and merges it.	- Valid write access to source repo (feature branch) - Settings which don't require additional review after commits	- Changes to source code w/o review - Changes to GitHub Actions workflow file w/o review	- Dismiss approvals after new changes are commited			- Sign all commits - Audit Log	- Sign all commits - Audit Log
		Hard-to-find malicious changes to pass review (Hidden Backdoor)	An attacker commits seemingly valid but actually malicious changes to feature branch. Valid approvers overlook the malice and approve the changes.	- Valid write access to source repo (feature branch)	- Changes to source code - Changes to GitHub Actions workflow file	- Force 2 person review(branch protection, CODEOWNERS) - Static / dynamic analysis (including fuzzing) - SCA (Software Composition Analysis)			- Sign all commits - Audit Log	- Sign all commits - Audit Log
			Suppose there are two files. File A describes direct dependencies (eg: *package.json*) and file B describes resolved dependencies (eg: package-lock.json) which is automatically generated and hard to read for humans. An attacker adds valid changes to A, and updates B. An attacker also adds modification to B to include malicious dependencies. Reviewers check only A and ignores maliciously modified B.	- Valid write access to source repo (feature branch)	- Changs to source code	- Force 2 person review(branch protection, CODEOWNERS) - Static / dynamic analysis (including fuzzing) - SCA (Software Composition Analysis) - Compare auto-generated files to its original files			- SBOM - Sign all commits - Audit Log	- SBOM - Sign all commits - Audit Log
		Bypass review (using exception of review strategy such as *`[skip XXX]`* commit message)	Suppose a repo has an exception in case PRs don't require approvals (eg: tiny fix of typo). An attacker commits changes which matches this exception and merges them without review.	- Valid write access to source repo (feature branch) - Existence of review exception (GitHub Actions, BOT, etc)	- Changes to source code w/o review - Changes to GitHub Actions workflow file w/o review	- Disallow exceptions for skippable PRs				
		Abuse bot accounts	Suppose a repo has an automated BOT account, which can automatically commits changes without review. An attacker compromises or use the BOT to commit malicious changes.	- Valid write access to source repo - Existence of vulnerable BOT	- Changes to source code w/o review - Changes to GitHub Actions workflow file w/o review	- Require admin's approval for any bots/apps to be installed (Allowlisting) - Grant only least permission to bots/apps			- Audit Log	- Audit Log
		Admin hijacking of repositories	An attacker compromises admin's account, then changes branch protection rule of a repository. Then attacker commits malicious changes using the admin account.	- Compromise of admin accounts	- Changes to source code w/o review - Changes to GitHub Actions workflow file w/o review	- Force 2FA - Limit the number of admins - Network restriction to source repo			- Audit Log	- Audit Log
CI (Process)	**C: Source Repository and GitHub Actions**	Malicious changes to code by CI tools	Suppose a repository has a CI tool (eg: linter), which can apply changes without review. An attacker can abuse this tools to commit malicious changes.	- Valid write access to source repo - Bot/tools have write access to source repo	- Changes to source code - Changes to GitHub Actions workflow file	- Grant least permission to Bot/tools				
		Use of unauthorized code (branch, tag)	Suppose that external resources are used as dependencies in a CI process, and references to those resources are mutable. An attacker can abuse this mutability to make GitHub Actions refer to unauthorized resources.	- Valid write access to source repo - Control over GitHub Actions workflow file / Permission to edit dependencies	- Control over artifact (binary injection, malformed behavior)	- Force 2 person review(branch protection, CODEOWNERS) - Use immutable reference to source (ref, branch, repo name)	- SBOM policy verification - Workflow attestation verification			- SBOM - Workflow attestation
			An attacker takes over a deprecated and removed GitHub repository abusing repository redirect. Then CI/CD uses unauthorized dependencies.	- Hijacking of GitHub username	- Control over artifact - Secret leakage	- Clone dependencies' repositories beforehand. - Network restriction				
		Use of modified GitHub Actions workflow file	An attacker modifies GitHub Actions definition file and commits changes. CI/CD are perfomed based on this modified workflow file.	- Valid write access to source repo	- Control over GitHub Actions (leading to secret leakage, control over artifact, etc)	- Force 2 person review(branch protection, CODEOWNERS) - Proper settings of triggers of CI	- Verification of workflow attestation	- Limit secrets passed to CI steps - Grant least permissions to CI steps - Short-lived and scoped secret - Rotation of keys		- Workflow attestation
		Bypass / Skip CI	Suppose the source repository platform allows to bypass/skip the CI process in certain circumstances. An attacker abuses this exception to skip CI requirements to merge malicious code.	- Valid write access to source repo - GitHub settings which contains exceptions about CI skip	- Changes to source code w/o CI test	- Disallow exceptions where CI can be skipped.	- Verification of workflow attestation			- Workflow attestation
			An attacker edits GitHub Actions workflow file to add a `paths` condition, which skips CI when certain files are not changed.	- Valid write access to source repo - GitHub settings which contains exceptions about CI skip	- Changes to source code w/o CI test	- Disallow skip-CI exceptions - Force 2 person review(branch protection, CODEOWNERS)				
		Use of malicious inputs (branch name, comment, commit message)	Suppose GitHub Actions workflow file accept external inputs (such as repo name, branch name, .PR name, comments...). An attacker abuses these inputs to invoke malicious actions.	- Valid write access to source repo / Valid permission to manipulate external inputs (comment etc)	N/A	- Do not take external inputs - Do not pass secrets to unverified steps - Verfy signature of external tools before execution - Hermetic Build / Network restriction (explicitly separate installation of dependencies and build)	- Verification of workflow attestation	- Limit secrets passed to CI steps - Grant least permissions to CI steps - Short-lived and scoped secret - Rotation of keys		- Workflow attestation
			An attacker adds malicious changes to test code, which is executed in test steps in CI.	- Valid write access to source repo	N/A	- 2 person review - Do not pass secrets to unverified steps - Verfy signature of external tools before execution - Hermetic Build / Network restriction (explicitly separate installation of dependencies and build)				
	C: Build	Unauthorized build process (unauthorized entry point, binary injection, ...)	An attacker modifies GitHub Actions flow to skip some of build process, or to include malicious dependencies.	- Valid write access to GitHub Actions actions	- Control over build process, leading to control over artifact	- Force 2 person review(branch protection, CODEOWNERS) - Centralized (standardized) build process, which cannot be modified by developers	- Verification of workflow attestation	- Limit secrets passed to CI steps - Grant least permissions to CI steps - Short-lived and scoped secret - Rotation of keys		- Workflow attestation
			Suppose a CI environment is affected by malware. Malware watches CI build process and injects malicious binary into built artifact.	- Compromise of CI environment	- Control over artifact	- Ephemeral build environment				

		Invalid interaction between subsequent build process	An attacker compromises at least one step of build flow to affect subsequent build process.	- Valid write access to GitHub Actions workflow file	- Control over build process, leading to control over artifact	- Ephemeral build environment - Immutable references to source repo or dependencies	- Verification of workflow attestation			- Workflow attestation
		Invalid interaction between parallel build process	An attacker compromises at least one job of build flow to affect other job executed in parallel.	- Valid write access to GitHub Actions workflow file	- Control over build process, leading to control over artifact	- Ephemeral build environment - Immutable references to source repo or dependencies - Disallow interaction between steps executed in parallel	- Verification of workflow attestation			- Workflow attestation
		Use of malicious inputs (branch name, comment, commit message)	Suppose a GitHub Actions checkout to a certain commit of repo based on external inputs (such as repo name, branch name, PR title, etc...). An attacker abuses external inputs to make the action checkout to unauthorized commits).	- Valid write access permission of GitHub Actions	- Control over build process, leading to control over artifact	- Limit external inputs to build process - Immutable references to source repo or dependencies - Centralized (standardized) build process, which cannot be modified by developers	- Verification of workflow attestation			- Workflow attestation
		Developers of each microservices do not follow secure standard ways to build	Developers of each microservices accidentally / intentionally do not follow secure ways to build artifacts. As a result, artifacts of each services lacks necessary information and process such as SBOM, container scans, and attestations.	N/A	N/A	- Centralized (standardized) build process, which cannot be modified by developers	- Verification of workflow attestation - Verification of other attestations			- Workflow attestation
CI (Dependency)	**D: Secrets**	Access to secrets by modified code / dependencies	An attacker leaks secrets in `ENV` using compromised dependencies used in main branch.	- Compromise of CI tools / Valid write access to GitHub Actions	- Secret leak	- Isolate each GitHub Actions step's ENV - Hermetic Build / Network restriction (explicitly separate installation of dependencies and build)		- Limit secrets passed to CI steps - Grant least permissions to CI steps - Short-lived and scoped secret - Rotation of keys		
		Access to secret by feature branches	An attacker leaks secrets in `ENV` using compromised dependencies used in feature branch. This might be possible even before merged to main branch.	- Compromise of CI tools / Valid write access to GitHub Actions - Access to prod secrets from CI of feature branch	- Secret leak	- Isolate each GitHub Actions step's ENV - Hermetic Build / Network restriction (explicitly separate installation of dependencies and build) - Separate secrets used in main and feature branch's CI		- Limit secrets passed to CI steps - Grant least permissions to CI steps - Short-lived and scoped secret - Rotation of keys		
	D: Dependency	Use of compromised dependencies	An attacker compromises a server where dependencies are distributed. An attacker then directly adds malicious changes to the packages. Victims don't notice it and uses the dependencies.	- Compromise of a server where dependencies are distributed	- Control over artifact	- Verify the integrity of dependency (iff hash is not compromised) - Lock version of dependencies - SCA (Software Composition Analysis) - Scoring of external dependencies (number of stars, age of tools, etc)	- SBOM policy		- SBOM	- SBOM
			An attacker adds a malicious changes to dependency's source repo, then they build and distribute an artifact. Victims don't notice it and uses the dependencies.	- Valid write access to dependency's source repo	- Control over artifact	- Lock version of dependencies - Clone dependency in domestic repo - SCA (Software Composition Analysis) - Scoring of external dependencies (number of stars, age of tools, etc)	- SBOM policy		- SBOM	- SBOM
			A valid owner of a dependency package adds malicious changes to their repository. Victims don't notice it and uses the dependencies.		- Control over artifact	N/A			- SBOM	- SBOM
		Use of unauthorized dependencies (including branch, tag)	An attacker modifies GitHub Actions workflow file to use unauthorized dependency (including branch and tag name).	- Valid write access to GitHub Actions workflow file	- Control over artifact	- Force 2 person review(branch protection, CODEOWNERS)	- SBOM policy		- SBOM	- SBOM
	Special Case	Sensitive data inside containers	Google Cloud Build includes all files of a workspace in an artifact unless .gitignore or .gcloudignore is provided. Developers mistakenly uploads sensitive data placed in a workspace, and publish it.		- Information leakage	- Provide appropriate .ignore files - Container scanning - Centralized (standardized) build process, which cannot be modified by developers	- Verification of attestations		- Attestation of container scanning result	- Attestation of container scanning result
Container Registry (OCI Registry)	**E: image push from *prod* CI to *prod* GCR**	Push unauthorized images	An attacker pushes unauthorized images from prod CI to prod GCR using compromised CI process.	- Control over build flow in prod CI	- Control over prod Kubernetes behaviour	- Centralized (standardized) build process, which cannot be modified by developers - Scoped permissions to push to GCR	- Verification of workflow attestation	- Scoped permissions to push	- Audit Log	- Workflow attestation - Audit Log
	E: image push from *dev* CI to *prod* GCR	Push unauthorized images	An attacker pushes unauthorized images built in dev CI to dev GCR using compromised CI process.	- Control over build flow in dev CI	- Control over dev Kubernetes behaviour	- Centralized (standardized) build process, which cannot be modified by developers - Scoped permissions to push to GCR	- Verification of workflow attestation (disallow feature branches as a signer)	- Scoped permissions to push	- Audit Log	- Workflow attestation - Audit Log
		Push to prod GCR from dev CI	An attacker pushes unauthorized images built in dev CI to prod GCR using compromised CI process.	- Control over build flow in dev CI - Push access permission from dev CI to GCR	- Control over prod Kubernetes behaviour	- Centralized (standardized) build process, which cannot be modified by developers - Scoped permissions to push to GCR	- Verification of workflow attestation (disallow feature branches as signer)	- Scoped permissions to push	- Audit Log	- Workflow attestation - Audit Log
	E: OCI Registry	Direct modification using stolen secrets / compromised accounts	An attacker directly pushes a malicious image to GCR using stolen credentials.	- Compromise of GCR accounts / Leakeage of GCR credentials	- Control over prod Kubernetes behaviour	- Limited and scoped access permission	- Verification of workflow attestation (disallow feature branches as signer)		- Audit Log	- Workflow attestation - Audit Log
CD	**F: CD pipeline**	Compromise Webhook to request to sync with arbitrary version of manifests	An attacker can request the deployment of Kubernetes manifest of arbitrary version using a stolen Webhook secret.	- Webhook secret (ArgoCD)	- Version of manifests applied to Kubernetes	- Use short-lived token (eg: OIDC token) - Verify that a specified reference of manifests is valid (repository name, branch name) - Disallow deployment of non-latest manifests.			- Audit Log	- Audit Log
		Apply changes to other services in a cluster	Suppose that ArgoCD has a strong permission over all services in Kubernetes cluster to apply manifests. An attacker can make ArgoCD apply changes to manifest of other services if service accounts and namespaces are not separated well.	- Valid write access to manifest source repo	- Edit of other services	- 2 person review - Separate Service Accounts of Argo by namespace (impersonation)				
	F: Production Cluster	Pull unauthorized image	An attacker modifies references to image sign Kubernets manifest. Victim cluster pulls an unauthorized image.	- Valid write access to Kubernetes manifests	N/A	- Verify integrity of images (signing) and its origin - 2 person review for manifest repository - Verify image policy (Attestations) - Egress restriction			- Audit Log	- Audit Log
		Pul compromised image	An attacker compromises images in an Container registry. Victim cluster pulls this compromised image.	- Write access to Container registry	N/A	- Verify integrity of images (signing) - Verify image policy (Attestations) - Egress restriction			- Audit Log	- Audit Log
		Direct deployment by attackers	An attacker deploys arbitrary services using compromised accounts.	- Compromise of Kubernetes accounts	N/A	- Appropriate RBAC - Verify integrity of images (signing) - Verify image policy (Attestations) - Verify manifest (eg: Manifest does not contain commands directly) - Egress restriction			- Audit Log	- Audit Log
										v1.01 (Dec 26, 2022)