Capital Investment Club

Secure SDLC and Change Management Policy

Repository, PR, testing, deployment, secrets, emergency changes, and security-sensitive change controls.

Last updated: 06/07/2026

Purpose

This policy defines CIC's secure engineering and change management requirements for building, testing, reviewing, deploying, and maintaining platform software.

Scope

This applies to CIC webapp, webapi, notificationservice, landingpage, infrastructure code, scripts, repositories, CI/CD workflows, production configuration, and supporting services.

Secure Development Principles

- Design for least privilege.
- Protect sensitive data.
- Validate inputs and outputs.
- Fail safely.
- Avoid secrets in code.
- Test business-critical logic.
- Review changes before merge.
- Preserve audit trail.
- Monitor production behavior.

Engineering teams and automation agents must treat business rules, authorization, data integrity, accessibility, and user experience as production requirements. A change is not complete merely because code compiles; it must preserve the user workflow and the underlying business invariants.

Change Types

- Standard change: routine, low-risk, tested change.
- Normal change: feature, fix, or configuration change requiring review.
- Emergency change: urgent fix for production impact or high-risk vulnerability.
- Security-sensitive change: affects auth, authorization, wallets, transactions, KYC-adjacent workflows,

production secrets, data exposure, or admin functions.

Requirements

Source Control

- Code must be maintained in approved repositories.
- Material changes must be made on branches.
- Pull requests must describe scope and testing.
- Sensitive changes must receive heightened review.

Branch and Pull Request Discipline

Material changes should use issue-linked branches. Pull requests should identify impacted workflows, tests run, screenshots or evidence for UI work, database or migration impact, rollback considerations, and any known limitations.

Pull requests for CIC product changes should also identify:

- Linked issue or approved request.
- User roles affected.
- API, database, schema, migration, or configuration impact.
- Authorization and privacy impact.
- Financial or transaction impact.
- Evidence location.
- Known residual risks.
- Whether production data repair or backfill is required.

Code Review

Code review must consider:

- Correctness
- Security
- Authorization
- Data integrity
- Input validation
- Error handling
- Logging
- Regression risk
- Test coverage

Reviewers should challenge assumptions and verify that the implementation matches the story, latest comments, and existing platform behavior. Comments from automated reviewers must be resolved or explicitly dispositioned before merge.

Testing

Material changes require appropriate tests:

- Unit tests
- Integration tests
- API tests
- Browser tests for user-facing flows
- Regression tests for previously broken behavior

Tests should be proportionate to risk. Financial, authorization, identity, wallet, grant, investment, public profile, support, and organization context changes require stronger tests than purely cosmetic changes.

Critical business flows require stronger validation. Examples:

- Wallet funding and disbursement
- Grant creation, publishing, joining, and contribution
- Pitch investment and equity calculations
- KYC-adjacent workflows
- Messaging and notifications
- Organization context switching
- Admin actions

Testing must include negative and edge cases, not just the happy path. For UI changes, agents should validate the rendered page in a browser and inspect whether controls are usable, labels are clear, buttons have appropriate contrast, responsive layouts are not broken, and loading states do not expose stale or unauthorized data.

Required Browser/UAT Evidence

For user-facing changes, evidence should include:

- Browser screenshot or video for the main flow.
- Negative scenario screenshot or result where applicable.
- Responsive or mobile evidence for layout-sensitive pages.
- Confirmation that primary buttons, links, inputs, dropdowns, uploads, and disabled states behave as labeled.
- Confirmation that user/organization context switching does not leak stale data.
- Console/network error review for the tested flow.

Business Rule Tests

Business-critical logic must have regression tests for the rule, not only component rendering. Examples include equity calculations, wallet movement timing, grant funding, co-grantor invitations, anonymous grantor display, organization-member grant application restrictions, assessment pricing, public profile visibility, KYC status display, and PFS ledger posting.

Secrets Management

- Secrets must not be committed to repositories.
- Secrets must be stored in approved secret management or environment configuration.
- Exposed secrets must be rotated.
- Production secrets must have restricted access.

CI/CD workflows must avoid printing secrets or deploying local key files. Ignore files and deployment packaging

rules must explicitly exclude local service-account keys, private env files, and other credentials.

Deployment

Deployments must be traceable to approved code and configuration. Failed deployments must be investigated before repeated attempts.

Deployment failures should be treated as defects until the actual logs are reviewed. A deployment must not be repeatedly rerun without understanding the current failure mode. Runtime, dependency, packaging, environment-variable, and secret issues must be fixed in source control or approved environment configuration.

Database and Migration Changes

Database or data repair changes must include:

- Scope of data affected.
- Backfill or migration strategy.
- Idempotency plan.
- Rollback or recovery plan where practical.
- Validation query or script.
- Evidence that historical records are handled correctly.

One-time data repair scripts must be idempotent where practical, logged, and documented with the target environment, affected records, dry-run output where available, execution output, and validation query.

UI/UX Validation

User-facing changes should be browser-tested. Validation should include desktop and mobile where the workflow is responsive, and should verify labels, button contrast, hover/focus states, error handling, empty states, and completion paths.

UI work should preserve existing functionality unless the story explicitly removes it. Redesign work must compare the rendered implementation against the approved mock or proposal and iterate until key layout, content, controls, and responsive behavior match.

Security Review Triggers

Security review is required for changes affecting authentication, authorization, wallet or transaction logic, financial calculations, user privacy, public profiles, file upload, support access, KYC-adjacent features, secrets, logging, or production infrastructure.

Security review is also required for changes involving PFS, ledger posting, financial API integrations, webhook handling, data export/download, support screen sharing, report-profile workflows, and organization context switching.

Emergency Changes

Emergency changes must be documented after the fact with:

- Reason
- Approver
- Change made

- Validation
- Follow-up remediation

Emergency changes must be reviewed after deployment to confirm no broader regression was introduced. Any temporary bypass, manual data fix, or reduced control must have an owner and expiration date.

Release Readiness Gate

A material change should not be considered ready to merge unless:

- The story/latest comments were reviewed.
- Affected code paths and adjacent flows were inspected.
- Automated tests were run or a justified exception is documented.
- Browser validation was performed for user-facing flows.
- Data and authorization impact were considered.
- Deployment and rollback considerations are documented for high-risk changes.
- Evidence is attached or linked where required.
- Known gaps are either fixed or explicitly accepted by the owner.

Post-Deployment Verification

After deployment of high-risk or user-facing changes, CIC should verify the deployed environment, not only local behavior. Verification should confirm the route loads, primary workflow works, and no obvious console, API, or permissions errors appear.

Change Evidence

Required evidence:

- Pull request
- Review record
- Test results
- Deployment record
- Issue or change ticket
- Rollback plan for high-risk changes where practical
- Browser/UAT evidence for user-facing changes
- Data repair or migration evidence where applicable
- Deployment verification evidence for high-risk changes

Vulnerability Fixes

Security fixes must be prioritized based on severity and tracked to closure.

Dependency vulnerability fixes must include lockfile updates, local validation with the package manager version used by CI, and deployment validation where cloud buildpack or platform scanners enforce runtime package versions.

Review

This policy must be reviewed at least annually.