kiuwan
by Sembi

SDLC AppSec Implementation Checklist (2026)

A practical checklist to implement SAST + SCA controls before deployment—so security scales with development velocity.

How to use this checklist

Purpose Implement and verify SDLC controls that reduce risk before code reaches production through continuous SAST + SCA scanning, prioritization, and measurable remediation.

Who it's for Engineering leaders, DevOps, AppSec consultants, and tech leads responsible for SDLC readiness.

How to complete Only check items that are implemented + verifiable (pipeline config, scan history, PR checks, tickets, reports).

Cadence Use during onboarding of new apps, before major releases, and quarterly for priority apps.

1 SAST + SCA BASELINE IN CI/CD
(Before deployment by default)

Goal: Ensure every change is evaluated with SAST + SCA as part of delivery.

Requirements:

- [ ] SAST runs automatically on Pull Requests (or build) for in-scope repositories
- [ ] SCA runs automatically to detect vulnerable dependencies (incl. transitive)
- [ ] Results appear where developers work (PR status checks / CI output)
- [ ] Scheduled scans exist (nightly/weekly) for full-coverage + drift detection
- [ ] A "minimum scan standard" is documented (when it runs + who owns it)

Implementation notes: Start with your top 3 apps, then template the pipeline configurations for rollout.

Completion criteria (verifiable): SAST+SCA run automatically and produce visible results for priority repositories, with scan history proving consistency.

2 POLICY & QUALITY GATES
(Progressive enforcement, not friction)

Goal: Enforce the right security gates **without slowing delivery**.

Requirements:

- ☐ A progressive enforcement model (Report-only → Block Critical → Expand)
- ☐ Clear thresholds exist for “fail vs warn” (e.g., Critical/High only at first)
- ☐ “New issues only” policy is defined (avoid punishing legacy backlog initially)
- ☐ A process for false positives (owner + response time)
- ☐ An exception list is produced, time-bound and tracked (no permanent bypass)

Implementation notes: Progressive gates improve adoption; over-blocking causes tool rejection.

Completion criteria (verifiable): At least one repo has enforced gates with documented thresholds and an exception/false-positive workflow.

3 PRIORITIZATION & REMEDIATION WORKFLOW
(Findings and fixes)

Goal: Turn scan findings into actionable items to remediate ownership + cadence.

Requirements:

- ☐ A simple priority model exists (P0/P1/P2) using **severity + exposure + context**
- ☐ Tiered SLA model exists (e.g. P0 = 48hr triage then 7-30 days fix [varies by patch availability], P1 = 60 days, P2 = 90 days or next release)
- ☐ Owners are assigned by repo/app/team (not “Security owns everything”)
- ☐ Findings are linked to work tracking (tickets/PRs) with closure evidence
- ☐ A recurring “Top risks” review happens for priority apps (weekly/biweekly)

Implementation notes: Leaders should optimize this section to **focus** on top 10 risks, not vulnerability volume.

Completion criteria (verifiable): Critical findings have owners + SLAs, and closure is visible in tickets/PRs with a reduction trend.

4 OPEN-SOURCE & DEPENDENCY RISK
(SCA in practice)

Goal: Reduce dependency-driven risk before shipping.

Requirements:

- [ ] Dependency inventory list for priority apps (including transitive dependencies)
- [ ] Transitive dependency risk model (e.g. prioritizes based on reachability/exploitability, defines when to upgrade direct dependencies vs accept risk, includes language-specific mitigation strategies)
- [ ] Vulnerable/deprecated dependencies are automatically detected and tracked
- [ ] License/compliance review is defined for priority repos (basic acceptable-use rules)
- [ ] Update cadence exists (monthly or per release cycle; not “when we remember”)
- [ ] A “high-risk dependency” policy exists (block or replace when Critical)

Implementation notes: Make dependency updates a habit; small, frequent updates reduce risk.

Completion criteria (verifiable): SCA findings are visible, owned, and decreasing over time; dependency update cadence is followed.

5 CONTAINER & INFRASTRUCTURE SECURITY

Goal: SDLC security to container images & IAC.

Requirements:

- [ ] Container image scanning integrated
- [ ] Base image update cadence defined (monthly minimum, critical CVEs trigger immediate rebuild)
- [ ] IaC scanning enabled (Terraform, CloudFormation, Kubernetes manifests scanned in CI/CD)
- [ ] Golden base image catalog exists
- [ ] Container security policies enforced (non-root users, read-only filesystems where applicable)

Implementation notes: Make container and IaC security part of your release checklist: Scan by default, standardize base images, and enforce a few high-impact policies first.

Completion criteria (verifiable): Automated scans + documented golden images + enforced policies + remediation proof.

6 AI-ASSISTED CODE OVERSIGHT
(Code + dependency scope)

Goal: Ensure AI-assisted development doesn't bypass your SAST/SCA standards.

Requirements:

- ☐ Document all AI-assisted coding usage at a minimum (what tools, where used)
- ☐ AI-assisted code follows the same PR review + SAST gates as any code
- ☐ Dependency name validation process exists (e.g. AI may suggest non-existent packages, creating typo risk)
- ☐ A lightweight rule exists: "AI output is treated like third-party code"
- ☐ Teams know where AI use is allowed vs restricted (simple internal guideline)
- ☐ High-risk library additions require heightened review (e.g. auth, networking libraries need human verification)

Implementation notes: Keep it lightweight and code-focused—this is about consistent SDLC standards.

Completion criteria (verifiable): AI-assisted code is not a special case scenario, it's reviewed, scanned, and gated like everything else.

7 REPORTING, MEETING COMPLIANCE & LEADERSHIP VISIBILITY

Goal: Make AppSec progress measurable and audit-ready without chaos.

Requirements:

- ☐ Standard reporting exists for priority apps (scan history + current top risks)
- ☐ Leadership metrics are defined and reviewed monthly:
  - Critical issues open
  - Time-to-remediate critical
  - Risk trend by application/repository
- ☐ Evidence artifacts stored centrally with documented retrieval process (scan reports, policies, tickets, ownership records with location and access defined)
- ☐ A quarterly posture review exists for high-risk apps (Security + Engineering)
- ☐ One accountable program owner exists (operating model + rollout)

(continued...)

Implementation notes: Leaders care about trendlines and confidence, not perfect scores.

Completion criteria (verifiable): You can show scan coverage, remediation progress, and trend reporting in under 2 hours.

Choose Your Action Plan

(Choose the path that matches your team's next 2 weeks)

- [ ] **Path A – Baseline (fastest):** Run SAST + SCA scans on your top 3 apps this week
- [ ] **Path B – Pipeline (most scalable):** Implement PR scanning + progressive gates in 1 repo in 2 weeks
- [ ] **Path C – Remediation (most visible):** Define priority model + SLAs + owners, then execute Top 10 fixes

If this checklist revealed any gaps, a 20-day Kiuwan trial can help you validate results against your **actual code and dependencies**—before deployment.

Learn More About Kiuwan:
kiuwan.com