SECURITY PROTOCOL — V1

The AI-Era Code Security Checklist

Engineering reference
For dev, review & production hand-off

AI-generated and AI-assisted code does not know the difference between a staging and a production environment — that judgment sits entirely with process, review, and infrastructure. This checklist covers four stages of the pipeline where key and credential leaks most commonly originate, plus the tooling to enforce each stage automatically rather than relying on memory.

01 WHILE CODING — KEY HYGIENE

- ☐ Never use production credentials during development — issue scoped, rate-limited dev keys per environment
- ☐ All secrets live in environment variables — never hardcoded in source, config files, or comments
- ☐ Separate keys per environment (dev / staging / prod), each independently rotatable and revocable
- ☐ Set spend caps, rate limits, and usage quotas on every key at the point of creation
- ☐ Prefer short-lived tokens — OAuth, STS, IAM role assumption — over long-lived static keys where supported
- ☐ Use a secrets manager beyond solo dev — Vault, AWS Secrets Manager, Doppler, or 1Password
- ☐ Never paste real credentials into AI prompts, even as "examples" — use placeholder values only
- ☐ Never expose LLM/third-party API keys (OpenAI, Anthropic, etc.) client-side — route all AI calls through a backend proxy, never embed keys in frontend bundles
- ☐ Verify AI-suggested packages actually exist and are legitimate before installing — AI models can hallucinate plausible but nonexistent or malicious package names ("slopsquatting")

02 PRE-PUSH — LOCAL VERIFICATION

- ☐ Install a secret scanner as a pre-commit hook — Gitleaks or detect-secrets — to block commits containing secrets
- ☐ Add a pre-push hook that re-scans the full diff before it ever leaves your machine
- ☐ Manually review the diff for hardcoded URLs, internal IPs, tokens, and leftover debug flags
- ☐ Run a dependency audit before pushing new packages — npm audit, pip-audit, or safety check
- ☐ Confirm sensitive files are gitignored — verify with **git check-ignore .env**
- ☐ Strip debug endpoints, verbose logging, and test credentials before every push
- ☐ Require human review of AI-generated code before merge, with specific attention to auth checks, input validation, and access-control logic — where AI most commonly gets it subtly wrong
- ☐ Restrict the AI coding agent's own file-system and shell access — no blanket read access to .env/secrets, no unattended shell execution for agentic tools (Claude Code, Cursor agent mode, Copilot workspace, etc.)

03 GIT CHECK — REPOSITORY HYGIENE

- ☐ Client and production repos are private by default — never public, even temporarily
- ☐ Enable branch protection on main / production — no direct pushes, PR review required to merge
- ☐ Turn on native secret scanning and push protection at the org or repo level
- ■ If a secret was ever committed: rotate it immediately, then purge history with git filter-repo or BFG — deletion alone does not remove it
- ☐ Restrict collaborator permissions to least privilege; audit repo access on a quarterly basis
- ☐ Require signed or verified commits on production-critical repositories
- ☐ Keep CI/CD secrets separate from local/dev secrets and scoped to the pipeline only — restrict which branches can trigger deploys

04 POST-LIVE — PRODUCTION OPERATIONS

- ☐ Rotate all production keys on a fixed schedule — 30 to 90 days depending on sensitivity
- ☐ Apply least-privilege IAM policies — scope every credential to exactly what it needs, nothing more
- ☐ Set hard billing caps and real-time spend alerts on every cloud and API account
- ☐ Configure anomaly alerts — a sudden spike in API calls is often the first sign of a leaked key
- ☐ Run a SAST or full security audit before go-live, and again after major releases
- ☐ Rate-limit and WAF-protect all public-facing endpoints
- ☐ Maintain an incident response plan — know how to revoke a compromised key in under 5 minutes
- ☐ Verify webhook signatures on all inbound integrations (Stripe, etc.) — a common gap when AI scaffolds a webhook handler without adding signature checks

RECOMMENDED TOOLS — FREE TIER

enforce each stage automatically, not from memory

Gitleaks	Pre-commit and pre-push secret scanning across the full codebase — the enforcement layer for section 01 and 02.	**Semgrep**	Free static analysis (SAST) that catches common vulnerability patterns beyond just leaked secrets, before go-live.
TruffleHog	Deep scan of full git history for credentials that were committed and later "removed" — catches what section 03 warns about.	**AWS Budgets / GCP Alerts**	Hard spend caps and real-time billing alerts — the last line of defense if a key does leak in production.
GitHub Push Protection	Native scanning that blocks known key patterns from ever being pushed, at the platform level, not just locally.	**Claude Security Review**	AI-powered audit of a codebase against a checklist like this one, run as a final pass before production hand-off.

NOTES / PROJECT-SPECIFIC EXCEPTIONS