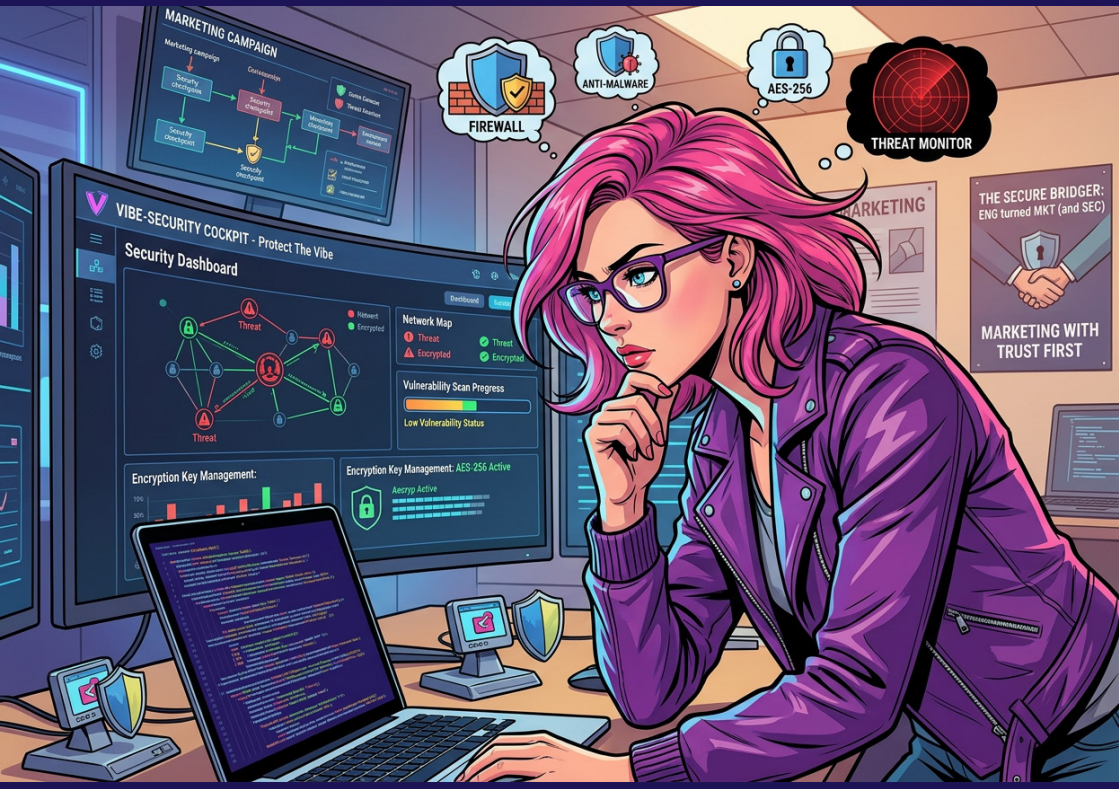
Vibecoding security checklist

Before You Ship:
The Minimum Viable Security Stack



cmimmio.com

SECTION 1: BEFORE YOU WRITE A SINGLE FEATURE

[ ] Set a security system prompt in your vibecoding tool before starting any build
[ ] Require no hardcoded credentials in your system prompt
[ ] Require parameterised database queries in your system prompt
[ ] Require input validation on all user-facing fields in your system prompt
[ ] Require rate limiting on authentication endpoints in your system prompt
[ ] Require standard security headers in your system prompt
[ ] Require secrets stored in environment variables in your system prompt
[ ] Create a .env file and add it to .gitignore before pushing anything to a repository
[ ] Enable Dependabot in your GitHub repository settings

SECTION 2: DURING EVERY FEATURE BUILD

After the AI generates any feature, run at least one security follow-up prompt before accepting the output.
[ ] "What vibecoding security vulnerabilities might exist in this code? List each one and explain how to fix it."
[ ] "Check this against OWASP Top 10 best practices. What is missing?"
[ ] "Are there any hardcoded credentials, exposed API keys, or unvalidated inputs in this code?"
[ ] "What happens if a malicious user submits unexpected input here?"
[ ] "Is there any SSRF risk in this URL-fetching code?"
[ ] "Add all standard security headers: Content-Security-Policy, Strict-Transport-Security, X-Frame-Options, X-Content-Type-Options, Referrer-Policy."
[ ] "What are the security risks of this approach, and how will you avoid them?" (use this before implementation, not after)

SECTION 3: BEFORE YOU GO LIVE

[ ] Connect Aikido Security or Snyk to your repository
[ ] Run a full security scan before your first public user
[ ] Review your Supabase or database configuration and confirm public access is off
[ ] Confirm row-level security is enabled on your database
[ ] Confirm all API keys and secrets are in environment variables, not in source code
[ ] Confirm security headers are set and active, not just written in the files
[ ] Check your dependency list for unpinned or deprecated packages and update or pin all versions
[ ] Confirm password hashing uses bcrypt or equivalent
[ ] Confirm rate limiting is connected to the application and working
[ ] Publish a privacy policy and link it from your footer
[ ] Confirm data is stored in the EU region if you are selling to European buyers
[ ] Add a named contact for data protection enquiries

SECTION 4: ONGOING AFTER LAUNCH

[ ] Run a security scan every time you add a significant new feature
[ ] Review Dependabot alerts weekly
[ ] Act on critical and high severity alerts within 48 hours
[ ] Audit your subprocessor list every quarter
[ ] Update your subprocessor list when you add new third-party tools
[ ] Run security follow-up prompts any time you use a new AI-generated code pattern
[ ] Keep a log of known issues and their fix status

SECTION 5: THE 7 RISKS TO CHECK FOR IN EVERY BUILD

[ ] Hardcoded credentials: no passwords, API keys, or tokens written directly into source code
[ ] Missing rate limiting: rate limiting is connected to the application, not just written in the files
[ ] No security headers: Content-Security-Policy, Strict-Transport-Security, and X-Frame-Options are present and active
[ ] Broken authentication: strong password hashing, secure token storage, and timing-attack protection are in place
[ ] Server-Side Request Forgery: URL-fetching features validate and restrict user-provided URLs
[ ] Vulnerable or hallucinated dependencies: all packages exist, are pinned to a version, and carry no known CVEs
[ ] Business logic vulnerabilities: authentication flows, access controls, and data handling have been manually reviewed, not just AI-generated and accepted

SECTION 6: TOOLS REFERENCE

[ ] Aikido Security connected: all-in-one SAST, SCA, DAST, secrets, containers, cloud. aikido.dev
[ ] Snyk connected: developer-first dependency and code security with IDE integration. snyk.io
[ ] Semgrep configured (optional): customisable static analysis for more technical teams. semgrep.io
[ ] SonarCloud connected (optional): code quality and security with GitHub pull request integration. sonarcloud.io
[ ] Dependabot enabled: automatic dependency vulnerability alerts and fix pull requests, built into GitHub, free

SECTION 7: SECURITY AS A SALES ARGUMENT

Be ready to answer these questions in European B2B procurement conversations:
[ ] Where is customer data stored? (answer: EU region)
[ ] Are you GDPR compliant? (answer: yes, with documentation to prove it)
[ ] Do you have a published privacy policy? (answer: yes, linked from footer)
[ ] Do you have a Data Processing Agreement available? (answer: yes, template ready on request)
[ ] Who is the named contact for data protection concerns? (answer: a real person with an email address)
[ ] Do you run automated security scanning? (answer: yes, and here is how)

SECTION 8: AI AGENT AND PROMPT INJECTION RISK

If your vibecoded product uses AI agents or processes external content, check the following:
[ ] Agents are granted least-privilege access only: they can access only what they absolutely need
[ ] All external inputs are validated before being passed to AI processing
[ ] Agent actions are logged so you can audit what happened if something goes wrong
[ ] No sensitive instructions or credentials are embedded in system prompts that external content could extract
[ ] You have tested what happens when a user submits unexpected or adversarial input to any AI-powered feature

This checklist is a practical starting point, not a complete legal or compliance framework. Consult a security professional and a lawyer for your specific product and market.

Notes

Notes

Secure you vibecoding!

If you feel you need more help, just contact C-Mimmi-O!

C-Mimmi-O website

C-Mimmi-O YouTube

C-Mimmi-O LinkedIn

C-Mimmi-O Facebook

C-Mimmi-O Instagram

C-Mimmi-O tumblr

cmimmio.com