Claude Code

audited.xyz
by zack.eth

Date
Apr 02, 2026 06:38 UTC

Commit
9f51e71
1,905 files, 514,008 lines

Summary

Claude Code is a TypeScript CLI application comprising approximately 514,000 lines of code across 1,905 files, functioning as an interactive AI coding assistant built on Anthropic's Claude model family. The application implements a sophisticated architecture spanning CLI entry points, a structured I/O protocol operating over multiple network transports including WebSocket, Server-Sent Events, and HTTP POST, an OAuth 2.0 authentication system with Proof Key for Code Exchange, an API client layer abstracting across Anthropic, AWS Bedrock, GCP Vertex, and Azure Foundry providers, and a React-based terminal user interface. The codebase handles sensitive operations including credential management, file system manipulation, and arbitrary code execution on behalf of the user, placing it in a high-trust security context.

The authentication subsystem demonstrates strong cryptographic practices, employing SHA-256 PKCE challenges derived from crypto.randomBytes(32), a localhost-bound OAuth callback server that mitigates redirect interception, and state parameter validation to prevent cross-site request forgery during the authorization flow. Input validation is enforced at API boundaries through Zod schema validation, and the file operation layer includes path traversal checks that prevent directory escape attacks. Additional defense-in-depth measures include HTML sanitization in error rendering paths, JavaScript line terminator escaping in NDJSON stream serialization to prevent injection across message boundaries, and bounded retry logic with abort signal propagation to prevent resource exhaustion under failure conditions.

This audit identified one Low-severity finding related to unrestricted environment variable mutation in the structured I/O transport layer. The processLine method accepts update_environment_variables messages and applies arbitrary key-value pairs to process.env without an allowlist, which in remote-worker configurations where messages traverse network transports could allow a compromised endpoint to inject security-sensitive variables such as HTTPS_PROXY or NODE_TLS_REJECT_UNAUTHORIZED, potentially enabling credential interception. The finding is mitigated by the fact that exploitation requires a compromised transport endpoint in a remote deployment topology, and the recommended fix—restricting mutations to CLAUDE_CODE_-prefixed keys—is narrowly scoped and low-risk to implement.

Findings

1 issues identified

1 Low

L-1 LOW Open

Unrestricted Environment Variable Mutation via Transport Messages

</> src/cli/structuredIO.ts

Description

The processLine method in StructuredIO accepts update_environment_variables messages and sets arbitrary process.env keys without an allowlist. In remote-worker configurations where messages traverse network transports, a compromised transport endpoint could inject security-sensitive environment variables (NODE_OPTIONS, PATH, LD_PRELOAD) to achieve code execution.

Impact

An attacker who can inject messages into the transport stream could achieve arbitrary code execution by setting NODE_OPTIONS, redirect file operations via HOME, or hijack subprocess execution via PATH. Requires compromising the transport layer or session ingress server.

Recommendation

Implement an allowlist restricting update_environment_variables to CLAUDE_CODE_* prefixed keys, which covers all documented use cases (session token refresh, environment runner version).

Fix

src/cli/structuredIO.ts

```
-       if (message.type === 'update_environment_variables') {
-         // Apply environment variable updates directly to process.env.
-         // Used by bridge session runner for auth token refresh
-         // (CLAUDE_CODE_SESSION_ACCESS_TOKEN) which must be readable
-         // by the REPL process itself, not just child Bash commands.
-         const keys = Object.keys(message.variables)
-         for (const [key, value] of Object.entries(message.variables)) {
-           process.env[key] = value
-         }
+       if (message.type === 'update_environment_variables') {
+         // Apply environment variable updates directly to process.env.
+         // Used by bridge session runner for auth token refresh
+         // (CLAUDE_CODE_SESSION_ACCESS_TOKEN) which must be readable
+         // by the REPL process itself, not just child Bash commands.
+         // Allowlist: only CLAUDE_CODE_* prefixed vars may be updated
+         // via transport to limit blast radius of a compromised transport.
+         const keys = Object.keys(message.variables)
```

```
+         for (const [key, value] of Object.entries(message.variables)) {
+           if (key.startsWith('CLAUDE_CODE_')) {
+             process.env[key] = value
+           } else {
+             logForDebugging(
+               `[structuredIO] rejected update_environment_variables key: ${key} (not in allowlist)`,
+               { level: 'warn' },
+             )
+           }
+         }
```

Conclusion

The Claude Code codebase exhibits a mature security posture consistent with a production application handling sensitive credentials and privileged system operations. The authentication implementation follows current best practices for public OAuth clients, the input validation layer is systematically applied at trust boundaries rather than ad hoc, and the serialization layer accounts for subtle injection vectors such as JavaScript line terminators in JSON streams. The consistent application of defense-in-depth patterns across disparate subsystems—from cryptographic token generation to file system access controls to network retry semantics—indicates that security considerations are integrated into the development process rather than applied as an afterthought.

The single Low-severity finding in the structured I/O transport layer represents an environment variable injection vector that is constrained in practice by its dependency on a compromised transport endpoint in remote-worker topologies. The attack surface is real but narrow: exploitation requires man-in-the-middle positioning on the transport channel and targets the runtime environment rather than application logic directly. The recommended remediation of applying a CLAUDE_CODE_* prefix allowlist to environment variable updates eliminates the attack vector with minimal implementation risk and no impact on legitimate functionality, as application-layer environment coordination does not require mutation of arbitrary process-level variables.

The overall security architecture of Claude Code is well-suited for its deployment context as a developer tool operating with elevated local privileges. The codebase demonstrates disciplined handling of the trust boundaries that matter most—authentication credential flows, file system access mediation, and cross-process message serialization—and the single finding identified does not represent a systemic weakness but rather a localized gap in input validation on a specific message type. With the recommended allowlist remediation applied, the application presents a strong security profile across its attack surface.