Third-Party Software Security Review — ScorpioX Code

Product: ScorpioX Code **Website:** https://code.scorpiox.net **Publisher:** ScorpioX Inc **Category:** AI-Powered Development Tool / CLI Platform **Language:** Pure C (zero external dependencies) **Audit Date:** 2026-04-28 **Codebase Commit:** `f7f14b49ca77bfb07d31a819de0791445d3c02a5` **Branch:** main **Classification:** CONFIDENTIAL — For Corporate Review Only

---

1. Executive Summary

This report presents the findings of a comprehensive security assessment of **ScorpioX Code**, an AI-powered development tool and CLI platform published by ScorpioX Inc. The product is built entirely in pure C (~370,000 lines of code) with zero package manager dependencies. All third-party libraries (mbedTLS for cryptography, yyjson for JSON parsing) are vendored as source code and compiled directly into the binary.

Key Security Strengths

- **Zero dependency supply chain**: No npm, pip, cargo, go modules, or any package manager is used. The C build has no external fetch steps — no `FetchContent`, no `ExternalProject`, no `git submodules`.
- **Single static binary**: On Linux, the application compiles to a fully statically-linked binary with only libc as a runtime dependency, eliminating shared library hijacking risks.
- **Full code provenance**: All 996 commits in the repository originate from ScorpioX organization email addresses (`scorpiox.net`, `scorpioplayer.com`). No external contributors were found.
- **Vendored source code**: The only two third-party libraries (mbedTLS ~225K LOC, yyjson ~743KB) are committed as auditable source with their original licenses.
- **No classic C vulnerabilities**: Zero instances of `strcpy`, `strcat`, `sprintf`, or `gets` in project-owned code. `snprintf` is used consistently (2,282 instances).

Key Findings Requiring Attention

Severity	Count	Summary
Critical	5	Hardcoded API keys in WASM config, SSL verification disabled on token fetchers, OAuth over plain HTTP, command injection in server CGI, SSH passwords in process listing
High	12	Command injection surfaces (web search, container, WSL, WhatsApp), raw TCP token transport, auto-update without code signing, plaintext credentials in config, unsalted password hashing
Medium	14	Stack buffer sizes, TOCTOU races, FRP weak crypto (MD5/64-iteration PBKDF2), missing error checks, predictable temp files, config

		masking gaps
Low	6	Missing compiler warnings, dead code, unused variables, signal handling
Info	3	Large stack buffers, Node.js bridge package.json (isolated), signal safety

Overall Risk Rating: MEDIUM

The core architecture demonstrates **exemplary supply chain security** — the zero-dependency, single-binary, pure-C approach with full commit provenance is best-in-class. However, the application has a broad attack surface due to its multi-tool nature (HTTP server, email server, DNS server, container runtime, VM hypervisor) and contains several areas where input validation and transport security should be hardened. The critical findings are primarily in peripheral components (WASM config, token fetchers, CGI server) rather than the core AI assistant workflow.

For corporate deployment as an AI coding assistant: The primary user-facing workflow (TUI → AI provider API) uses HTTPS with proper TLS via libcurl and represents **low risk**. The critical findings affect infrastructure/deployment components that may not be relevant to all deployment scenarios.

---

2. Application Overview

ScorpioX Code (https://code.scorpiox.net) is an AI-powered coding assistant, agent orchestrator, and container runtime platform. It provides:

- **AI Chat TUI**: Interactive terminal interface supporting multiple AI providers (Anthropic Claude, OpenAI/Codex, Google Gemini) with streaming responses
- **Agent Orchestration**: SDK for orchestrating AI agents with tool use, file editing, and command execution
- **Container Runtime**: Linux container engine using user namespaces (`scorpiox-unshare`) with rootless operation by default
- **VM Hypervisor**: KVM-based virtual machine support (`scorpiox-vm`)
- **Development Tools**: Built-in web server, email server, DNS server, file transfer, voice transcription, and terminal multiplexer

Technical Architecture

Attribute	Detail
Language	C11 (pure C, no C++)
Lines of Code	369,495 total (143,599 first-party + 225,896 vendored)
Build System	CMake 3.16+
Compiler	System gcc/clang
Linking	Static by default (Linux)
Third-Party Libraries	mbedTLS (Apache-2.0/GPL-2.0+), yyjson (MIT) — both vendored as source
Package Dependencies	**None** — zero npm/pip/cargo/go/vcpkg/conan dependencies
Platforms	Linux, macOS, Windows, WASM

Output	~50+ individual executables, all statically linked

---

3. Architecture & Supply Chain

Source: 01-architecture.md

3.1 Dependency Model

ScorpioX Code follows a **vendored dependency model** — the gold standard for supply chain security:

Dependency	Type	Location	LOC	License
mbedTLS	Vendored source	`scorpiox/vendor/mbedtls/`	~225,000	Apache-2.0 / GPL-2.0+
yyjson	Vendored source	`scorpiox/vendor/yyjson/`	~743KB (2 files)	MIT
libcurl	System library	`find_package(CURL)`	N/A	MIT
pthreads	System library	`find_package(Threads)`	N/A	System

No package manager files affect the C build. A `bridge/package.json` exists for a standalone WhatsApp bridge (TypeScript/Bun) which is entirely separate from the C compilation pipeline.

3.2 Build Process Security

- **No network downloads during build**: Zero instances of `FetchContent`, `ExternalProject`, `file(DOWNLOAD)`, or URL-based downloads in CMake
- **Static linking**: `SX_STATIC_LINK ON` by default — eliminates shared library hijacking
- **Frozen code generation**: A Python script generates model definitions at build time but is frozen (no network fetch)
- **System compiler**: Uses system-installed gcc/clang — no downloaded/pinned toolchain

3.3 Code Provenance

Author Domain	Commits	Percentage
`scorpiox.net`	990	99.4%
`scorpioplayer.com`	5	0.5%
`sx@sx` (local alias)	1	0.1%

All 996 commits originate from ScorpioX organization email addresses. No external contributors, no public email providers (Gmail, Outlook, etc.).

3.4 Findings

Finding	Severity	Status	Recommendation
Pre-built ELF binaries in `bridge/` directory (2 files)	Medium	Acknowledged	Build from source in CI; remove pre-built binaries from repo
Missing compiler hardening flags (`-`			

`fstack-protector-strong`, `-D_FORTIFY_SOURCE=2`)	Medium	Open	Add to `CMAKE_C_FLAGS`
`curl | bash` in WhatsApp release script	Medium	Acknowledged	Replace with pinned, hash-verified download
`bridge/package.json` with Node.js deps	Info	Acknowledged	Isolated from C build; no action needed

---

4. Network Security

Source: 02-network.md

4.1 Network Activity Overview

ScorpioX Code is a **network-active** application with ~90+ source files containing network code. Network activity is categorized as:

Category	Protocol	User-Initiated	Purpose
AI Provider APIs	HTTPS (443)	✓	Core chat functionality (Anthropic, OpenAI, Gemini)
OAuth Token Refresh	HTTPS (443)	Automatic	Token lifecycle management
Token Fetch Services	HTTP/TCP	Automatic	API key retrieval from ScorpioX infrastructure
Session Telemetry	HTTPS (443)	Opt-in	Usage tracking (when `EMIT_SESSION_TRACKING=1`)
Distribution/Updates	HTTPS (443)	User action	Binary downloads, container images
Listening Services	TCP (various)	User action	HTTP server, DNS, SMTP, IMAP, WebSocket bridge

4.2 TLS Implementation

- **Primary**: libcurl handles all HTTPS connections with system CA certificates
- **Secondary**: mbedTLS provides TLS for SMTP, FRP tunneling, and mail queue
- **WASM**: Browser `fetch()` API inherits browser TLS

4.3 Critical Network Findings

Finding	Severity	Detail
Hardcoded API keys in WASM embedded config	**Critical**	`ANTHROPIC_ANTIGRAVITY_KEY` and `ANTHROPIC_ZAI_KEY` compiled into WASM binary; extractable
SSL verification disabled on 6+ token fetchers	**Critical**	`CURLOPT_SSL_VERIFYPEER=0` and `CURLOPT_SSL_VERIFYHOST=0` on OAuth/token fetch paths; MITM risk
OAuth tokens transported over	**Critical**	`token.scorpiox.net` endpoints use `http://` not

plain HTTP		`https://` ; tokens interceptable
Raw TCP token transport (no encryption)	High	`sx_tcp_fetch.c` sends `SXV1 AUTH=key` in plaintext over TCP
FRP tunnel TLS uses `VERIFY_NONE`	High	Tunnel endpoint identity not verified
Auto-update without code signing	High	`scorpiox-wsl.c` downloads and replaces binary without signature verification
Session telemetry posts full conversations	High	When enabled, complete prompts/responses sent to `code.scorpiox.net`
Hardcoded LAN IPs in config	High	`192.168.1.x` addresses expose internal network topology

4.4 Network Mitigations

- All AI provider API calls use HTTPS with proper TLS verification via libcurl
- OAuth refresh flows to `anthropic.com` and `openai.com` use HTTPS
- FRP tunneling encrypts payload with AES-128-CFB
- DNS server supports audit logging
- Session telemetry is opt-in (`EMIT_SESSION_TRACKING` defaults to off)

---

5. Data Handling & Privacy

Source: 03-data-handling.md

5.1 Data Flow Architecture

```
User Input → sx TUI → libsxnet (API provider) → HTTPS → Remote AI API
                ↓
       .scorpiox/sessions/<id>/   (conversations, traffic, logs)
       .scorpiox/conversations/   (persistent conversation JSON)
       .scorpiox/traces/          (debug trace JSONL)
```

5.2 Data Sensitivity by Component

Component	Sensitivity	Data Handled
`sx` (main TUI)	HIGH	API keys, user conversations, system prompts
`scorpiox-server`	HIGH	JWT secrets, POST bodies, auth headers
`scorpiox-server-email`	CRITICAL	Email content, user passwords, TLS keys, DKIM keys
`scorpiox-sshpass`	CRITICAL	Plaintext passwords via CLI `-p` flag
`scorpiox-otp`	HIGH	TOTP secrets via CLI `-s` flag
`libsxutil`	HIGH	Config cascade, logging, session management

5.3 Configuration & Secrets Management

Configuration uses a cascade model: `built-in defaults → exe_dir/scorpiox-env.txt → .scorpiox/scorpiox-env.txt → ~/.config/scorpiox/scorpiox-env.txt`. API keys and secrets are stored as plaintext in these config files.

Positive findings: - Config snapshots at session start redact sensitive keys (masked with `****`) - Config system (`sx_config_get()`) avoids polluting process environment with secrets - `scorpiox-sshpass` passes password via PTY write (not environment variable)

Negative findings: - No encryption at rest for any persistent data - All credentials stored as plaintext in config files - Session traffic directory contains full API payloads including headers with API keys - TOTP secrets and SSH passwords visible in `/proc/<pid>/cmdline`

5.4 Encryption Assessment

Mechanism	Implementation	Strength
HTTPS (AI APIs)	libcurl + system CA	✅ Strong
SMTP TLS	mbedTLS	⚠ Cert verification sometimes disabled
FRP tunnel	AES-128-CFB + PBKDF2	⚠ Only 64 PBKDF2 iterations; MD5 login hash
Email passwords	SHA-256 (unsalted)	❌ Weak — rainbow table vulnerable
Data at rest	None	❌ Not implemented

5.5 Data Retention

- Session retention: configurable via `SESSION_RETENTION_DAYS`; cleanup at session creation
- Conversation retention: no automatic cleanup; persists indefinitely
- Email retention: Maildir-based; no automatic expiration
- Default: sessions persist indefinitely if retention not configured

5.6 Critical Data Handling Findings

Finding	Severity	Impact
SSH passwords visible in process listing (`ps`)	Critical	Any user on the system can read SSH passwords
Email password hashing uses unsalted SHA-256	Critical	Rainbow table attacks on stolen accounts file
No encryption at rest for persistent data	Critical	Full data exposure if filesystem is compromised
All credentials in plaintext config files	High	Credential theft via file read access
Session traffic contains full API payloads	High	API keys in headers, full conversation content
Trace files contain unredacted API data	High	Debug traces expose sensitive request/response data
FRP PBKDF2 uses only 64 iterations	High	Brute-force key derivation feasible
Predictable temp file names (no mkstemp)	High	Symlink/TOCTOU attacks on `/tmp/sx_post_*`

---

6. Code Security Vulnerabilities

Source: 04-vulnerabilities.md

6.1 Vulnerability Overview

The codebase demonstrates **generally sound C programming practices**: - ✅ Zero instances of `strcpy`, `strcat`, `sprintf`, or `gets` - ✅ Consistent use of `snprintf` (2,282 instances) - ✅ `malloc`/`calloc` return values generally checked - ⚠ `system()`/`popen()` used extensively (118+ and 61+ call sites respectively)

6.2 Critical & High Vulnerabilities

ID	Severity	CVSS	Location	Description
V-01	**Critical**	9.8	`scorpiox-server.c:1568-1652`	CGI env vars embedded in Windows `cmd /C` via `system()` — sanitization strips only 6 characters
V-02	**High**	8.6	`scorpiox-websearch.c:222`	URL passed to `popen()` via `scorpiox-fetch "%s"` — shell metacharacter injection possible
V-03	**High**	8.1	`scorpiox-unshare.c:1245`	`which %s` via `system()` with package name from config — no input validation
V-04	**High**	7.5	`scorpiox-wsl.c:676-878`	`system()` with 16KB command buffer from user inputs, `wsl.exe -d %s`
V-05	**High**	7.5	`scorpiox-whatsapp.c:620-983`	Multiple `system()` calls with tmux commands from message data
V-06	**High**	7.4	`bridge/ws2tcp.c:504`	Path traversal check (`strstr(url_path, "..")`) bypassable via `%2e%2e` encoding

6.3 Medium Vulnerabilities

ID	Severity	CVSS	Location	Description
V-07	Medium	6.5	`sxmail_dkim.c:414`	32KB stack buffer — bounds checked but excessive
V-08	Medium	6.5	`scorpiox-wsl.c:676,721`	16KB + 12KB stack buffers risk stack overflow
V-09	Medium	6.2	`sxmux_session.c:1841`	Type confusion — comparison always false
V-10	Medium	5.9	`scorpiox-vi.c`	Unchecked `realloc` — original pointer lost on failure
V-11	Medium	5.5	118+ sites	Pervasive `snprintf(cmd) + system(cmd)` pattern
V-12	Medium	5.3	`scorpiox-server.c:897-914`	TOCTOU: `access()` → `realpath()` → `fopen()` race window

6.4 Positive Security Patterns

The codebase employs several commendable security practices: - **No classic buffer overflows**: Complete absence of `strcpy` / `strcat` / `sprintf` / `gets` - **Consistent bounds checking**: `snprintf` used throughout with proper buffer sizes - **Memory allocation checks**: `malloc` / `calloc` returns are generally validated - **Static linking**: Eliminates DLL/shared library hijacking vectors

---

7. Permissions & System Access

Source: 05-permissions.md

7.1 Permission Model Overview

The ScorpioX platform encompasses components with vastly different privilege requirements:

Risk Level	Component	Privilege Required	Reason
Critical	`scorpiox-traffic`	User	Disables TLS verification globally for child processes
Critical	`scorpiox-sshpass`	User	Disables all SSH host key checking by design
🟠 High	`scorpiox-unshare`	Root (privileged mode)	Container runtime with `--privileged` bypass
🟠 High	`scorpiox-vm`	`kvm` group	Direct `/dev/kvm` access, mmap of guest memory
🟠 High	`scorpiox-server`	User	Fork-per-request executing arbitrary Python scripts
🟠 High	`scorpiox-thunderbolt4`	Root	Raw BPF Ethernet frame access
🟡 Medium	`sx` (main TUI)	User	Extensive `system()` / `popen()` for subcommand dispatch
🟢 Low	`scorpiox-podman`	User	Thin wrapper delegating to Podman

7.2 Process Spawning

Mechanism	Count	Risk
`system()`	107	🟠 High — shell injection surface
`exec*()` family	66	🟡 Medium — direct exec
`popen()`	61	🟠 High — shell-mediated pipe
`fork()`	47	🟡 Medium — process isolation
`CreateProcessA()` (Windows)	40	🟡 Medium
`clone()` (Linux namespaces)	1	🟠 High — namespace creation

7.3 Sandboxing & Isolation

Container Runtime (`scorpiox-unshare`): - ✅ User namespaces by default (rootless containers) - ✅ Mount namespace isolation with pivot_root - ✅ PID namespace isolation - ✅ Network namespace (optional) - ✅ Cgroup v2 memory limits - ⚠ `--privileged` mode bypasses user namespaces - ❌ No seccomp filtering - ❌ No AppArmor/SELinux profiles

No setuid binaries: The codebase does not install any setuid/setgid binaries. It relies on system-provided setuid helpers (`newuidmap` / `newgidmap`) for container UID mapping.

7.4 Filesystem Access Patterns

- Configuration: reads from `exe_dir`, CWD `.scorpiox/`, and `~/.config/scorpiox/`
- Session data: writes to `.scorpiox/sessions/<id>/` in the working directory
- Container images: downloaded to and extracted in working directory
- Temp files: `/tmp/sx_post_*`, `/tmp/sxmux/` (predictable names)
- No access to files outside the working directory tree in normal operation

---

8. Consolidated Risk Matrix

#	Area	Finding	Severity	Status	Recommendation
1	Network	Hardcoded API keys in WASM embedded config	**Critical**	Open	Remove keys from source; use runtime config or environment variables for WASM builds
2	Network	SSL verification disabled on 6+ token fetchers	**Critical**	Open	Enable `CURLOPT_SSL_VERIFYPEER` and `CURLOPT_SSL_VERIFYHOST` on all HTTPS connections
3	Network	OAuth tokens transported over plain HTTP (`token.scorpiox.net`)	**Critical**	Open	Migrate all token endpoints to HTTPS
4	Vulnerabilities	CGI command injection in server (Windows `cmd /C`)	**Critical**	Open	Replace `system()` with `execve()`-based approach; implement allowlist-based input sanitization
5	Data	SSH passwords visible in process listing (`/proc/pid/cmdline`)	**Critical**	Open	Use stdin or file descriptor passing instead of CLI `-p` flag
6	Data	Email password hashing uses unsalted SHA-256	**Critical**	Open	Migrate to bcrypt/scrypt/Argon2 with per-user salts
7	Data	No encryption at rest for persistent data	**Critical**	Acknowledged	Implement optional at-rest encryption for session data and credentials
8	Vulnerabilities	Web search URL injection via `popen()`	**High**	Open	Sanitize URLs; use `exec*()` instead of `popen()` with shell
9	Vulnerabilities	Container runtime `which %s` injection	**High**	Open	Validate package names against allowlist
					Use `CreateProcessW()`

10	Vulnerabilities	WSL command injection (16KB buffer)	**High**	Open	with argument array instead of `system()`
11	Vulnerabilities	WhatsApp bridge command injection via message data	**High**	Open	Sanitize message content before shell interpolation
12	Vulnerabilities	Path traversal bypass in ws2tcp (`%2e%2e`)	**High**	Open	URL-decode before path traversal check; use `realpath()` validation
13	Network	Raw TCP token transport (plaintext API keys)	**High**	Open	Wrap TCP connections in TLS
14	Network	FRP tunnel TLS uses `VERIFY_NONE`	**High**	Open	Implement proper certificate verification or certificate pinning
15	Network	Auto-update without code signing	**High**	Open	Add Ed25519/GPG signature verification for downloaded binaries
16	Network	Session telemetry posts full conversations	**High**	Acknowledged	Document clearly; ensure opt-in only; consider anonymization
17	Data	All credentials stored as plaintext in config files	**High**	Acknowledged	Consider OS keychain integration or encrypted config
18	Data	Session traffic contains full API payloads with keys	**High**	Acknowledged	Redact Authorization headers in traffic captures
19	Data	FRP PBKDF2 uses only 64 iterations / MD5 login hash	**High**	Open	Increase to 600,000+ iterations; replace MD5 with SHA-256+
20	Data	Predictable temp file names (no `mkstemp`)	**High**	Open	Use `mkstemp()` for all temporary file creation
21	Permissions	`scorpiox-traffic` disables TLS verification globally	**High**	Acknowledged	Add warning banner; restrict to development environments
22	Permissions	`scorpiox-sshpass` disables SSH host key checking	**High**	Acknowledged	Make host key checking configurable rather than force-disabled
23	Architecture	Pre-built ELF binaries in `bridge/`	**Medium**	Acknowledged	Build from source in CI
24	Architecture	Missing compiler hardening flags	**Medium**	Open	Add `-fstack-protector-strong -D_FORTIFY_SOURCE=2`
25	Architecture	`curl | bash` in WhatsApp release script	**Medium**	Acknowledged	Use hash-verified download
26	Vulnerabilities	Large stack buffers (32KB in DKIM, 16KB in WSL)	**Medium**	Acknowledged	Move to heap allocation for buffers >4KB
27	Vulnerabilities	TOCTOU in server file serving	**Medium**	Open	Use `openat()` / `fstatat()` for atomic check-and-open
28	Vulnerabilities	Unchecked `realloc` (pointer loss on failure)	**Medium**	Open	Save original pointer before `realloc` ; handle NULL

					return
29	Vulnerabilities	Type confusion in sxmux_session (always-false compare)	**Medium**	Open	Fix data type or comparison logic
30	Vulnerabilities	Pervasive `snprintf + system()` pattern (118+ sites)	**Medium**	Acknowledged	Gradually migrate to `exec*()` family; create safe command builder API
31	Data	Config masking misses `*_PASS`, `*_SECRET` patterns	**Medium**	Open	Extend masking to all sensitive config key patterns
32	Data	Fetchtoken API key comparison not constant-time	**Medium**	Open	Use `CRYPTO_memcmp()` or equivalent
33	Network	Hardcoded LAN IPs in embedded config	**Medium**	Acknowledged	Move to runtime configuration
34	Network	DNS server on port 53 (spoofing risk)	**Medium**	Acknowledged	Document security implications; bind to localhost by default
35	Permissions	Container `--privileged` mode bypasses namespaces	**Medium**	Acknowledged	Require explicit flag; add warning
36	Permissions	No seccomp/AppArmor for containers	**Medium**	Open	Add seccomp BPF filter for container workloads
37	Vulnerabilities	Missing `fscanf` return value check	**Low**	Open	Check return value of `fscanf`
38	Vulnerabilities	Missing `#include` headers (33 compiler warnings)	**Low**	Open	Add missing standard library includes
39	Vulnerabilities	Dead code / unused variables	**Low**	Open	Remove or complete implementation
40	Vulnerabilities	`signal()` instead of `sigaction()`	**Info**	Acknowledged	Migrate to `sigaction()` for portable behavior

---

9. Conclusion & Recommendations

9.1 Overall Assessment

ScorpioX Code demonstrates a **security-conscious architecture** that is rare among modern development tools. The pure C implementation with zero package manager dependencies, vendored source-only libraries, static linking, and single-organization commit provenance eliminates entire categories of supply chain attacks that plague tools built on npm, pip, or other package ecosystems.

The codebase shows disciplined C programming with consistent use of `snprintf` and absence of classic buffer overflow patterns. The vendored dependency approach (mbedTLS + yyjson as source) provides full auditability and eliminates dependency confusion attacks.

However, the application's broad scope — encompassing an AI assistant, HTTP server, email server, DNS server, container runtime, and VM hypervisor — creates a wide attack surface. The most significant findings relate to:

1. **Transport security gaps**: Token fetchers and internal services that disable TLS verification or use plaintext protocols
2. **Command injection surfaces**: The pervasive `system()` / `popen()` pattern with shell

interpolation
3. **Secrets management**: Plaintext credential storage and process-visible passwords

9.2 Priority Recommendations

Immediate (Critical — fix before production deployment):

1. Remove hardcoded API keys from WASM embedded configuration
2. Enable TLS certificate verification on all token fetch and OAuth paths
3. Migrate `token.scorpiox.net` endpoints to HTTPS
4. Replace `system()` with `execve()` in network-reachable code paths (CGI server)
5. Stop passing passwords/secrets via command-line arguments (use stdin/FD)

Short-term (High — address within 30 days):

6. Implement code signing for auto-update downloads
7. Migrate email password storage to bcrypt/Argon2 with salts
8. Add compiler hardening flags (`-fstack-protector-strong` , `-D_FORTIFY_SOURCE=2`)
9. Increase FRP PBKDF2 iterations to ≥600,000; replace MD5 login
10. Use `mkstemp()` for all temporary file creation

Medium-term (Medium — address within 90 days):

11. Create a safe command builder API to gradually replace `snprintf + system()` patterns
12. Add seccomp BPF filtering for container workloads
13. Implement optional at-rest encryption for session data
14. Extend config masking to cover all sensitive key patterns
15. Add constant-time comparison for token/key validation

9.3 Suitability for Corporate Use

As an AI coding assistant (primary use case): ScorpioX Code's core workflow — user types a prompt, the TUI sends it to an AI API over HTTPS, and displays the response — is **low risk**. The HTTPS transport, proper TLS verification on AI provider connections, and absence of telemetry by default make it suitable for corporate environments where:

- The deployment uses API keys from the organization's own AI provider accounts
- The `scorpiox-env.txt` config file is protected with appropriate filesystem permissions
- Optional infrastructure components (email server, DNS server, container runtime) are evaluated separately if needed
- Session data directories are treated as sensitive and included in backup/retention policies

---

10. Appendix: Audit Methodology

10.1 Scope

This audit covered the complete ScorpioX Code codebase at commit `f7f14b49ca77bfb07d31a819de0791445d3c02a5` on the `main` branch. The audit encompassed 369,495 lines of code across 493 C/H source files.

10.2 Audit Agents

Five specialized security audit agents conducted independent analyses:

Agent	Report	Focus Area
Architecture Auditor	01-architecture.md	Build system, dependencies, supply chain, code provenance

Network Auditor	02-network.md	Endpoints, protocols, TLS, telemetry, DNS
Data Handling Auditor	03-data-handling.md	File I/O, secrets, encryption, privacy, data retention
Vulnerability Auditor	04-vulnerabilities.md	Buffer overflows, injection, memory safety, code patterns
Permissions Auditor	05-permissions.md	Privilege model, process spawning, sandboxing, filesystem access

10.3 Methodology

Each agent performed: - **Static analysis**: Source code review of all non-vendor C files - **Pattern matching**: Automated scanning for dangerous functions, hardcoded credentials, and insecure patterns - **Build system review**: CMake configuration, compiler flags, linking strategy - **Configuration analysis**: Config file formats, cascade model, secret handling - **Git history review**: Commit provenance, author analysis, binary file detection

10.4 Limitations

- This is a **static code audit** — no runtime testing, fuzzing, or penetration testing was performed
- Vendor code (mbedTLS, yyjson) was identified but not audited in depth (these are well-known, independently audited libraries)
- Windows-specific code paths (`CreateProcessA` , `cmd /C`) were reviewed statically but not tested on Windows
- WASM build configuration was reviewed but not compiled/tested

10.5 Classification

This report is classified as **CONFIDENTIAL — For Corporate Review Only**. Distribution should be limited to security teams, IT governance, and authorized decision-makers evaluating ScorpioX Code for corporate deployment.

Report compiled on 2026-04-28 by Security Audit Agent ScorpioX Code — https://code.scorpiox.net