Third-Party Software Security Review — ScorpioX Code

Software: ScorpioX Code
Type: AI-Powered Development Tool / CLI Platform
Language: Pure C (zero external dependencies)
Audit Date: 2026-04-28
Codebase Commit: `40e8525bd1b39a2512668945e5fbdddaf1101ccc`
Classification: CONFIDENTIAL — For Corporate Review Only

---

1. Executive Summary

This report presents a comprehensive security assessment of ScorpioX Code, an AI-powered development tool and CLI platform written in pure C. The audit was conducted by 13 specialized automated agents covering supply chain, build provenance, network security, TLS/SSL, credential management, file I/O, buffer safety, memory safety, command injection, privilege/access control, Windows deployment, telemetry/tracking, and install script security.

Key Metrics

Metric	Value
Total Lines of Code	380,496
Project Code	151,970 lines
Vendor Code	228,526 lines (mbedTLS, yyjson)
Source Files Audited	149 non-vendor C files
Findings (All Platforms)	212 total
Findings (Windows Deployment)	80 total
Overall Risk Rating	**MEDIUM**

Findings by Severity (All Platforms)

Severity	Count
Critical	9
High	20
Medium	33
Low	29
Informational	121
Total	**212**

Summary Assessment

The software demonstrates **strong foundational security practices** for a C codebase:

comprehensive compiler hardening flags, full static linking, zero use of `strcpy` / `sprintf` / `gets`, a safe-to-unsafe string function ratio of 266:1, and all telemetry disabled by default. The two vendored libraries (mbedTLS 3.6.6, yyjson 0.12.0) are well-known, maintained, and carry minimal supply chain risk.

However, several areas require attention:

- **Install & Distribution (CRITICAL):** The `curl|bash` install pattern lacks cryptographic code signing, and SHA256 verification degrades gracefully to no verification when checksum files are unavailable.
- **Command Injection (HIGH):** Three critical injection vectors exist in macOS-only iMessage integration and the interactive shell-out feature. Windows deployment is not affected by the iMessage findings.
- **Network Endpoints (HIGH):** A hardcoded Git PAT in a release script and plaintext HTTP for token-fetching endpoints.
- **File I/O (HIGH):** Predictable temp file paths in WASM/Linux code paths enable symlink attacks.
- **Memory Safety (MEDIUM):** 168 unchecked `strdup()` return values across 39 source files.

For **Windows workstation deployment** specifically, the risk profile is **LOW-MEDIUM** — the most severe findings (iMessage injection, Linux container symlink attacks, WASM temp files) do not apply to the Windows platform.

---

2. Deployment Scope — Windows Workstation

2.1 Primary Deployment: Windows Workstation

The following **56 binaries** are compiled for Windows (`.exe`):

Core Application

Binary	Purpose
`sx.exe` / `scorpiox.exe`	Main AI coding assistant TUI

Shell & Terminal Tools

Binary	Purpose
`scorpiox-bash.exe`	Bash command execution
`scorpiox-busybox.exe`	Unix coreutils for Windows
`scorpiox-pwsh.exe`	Remote PowerShell via REST
`scorpiox-wsl.exe`	WSL2 container management
`scorpiox-vi.exe`	Text editor
`scorpiox-tmux.exe`	Terminal multiplexer
`scorpiox-multiplexer.exe`	Session multiplexer

Configuration & Utilities

Binary	Purpose
`scorpiox-config.exe`	Configuration management
`scorpiox-conv.exe`	Format conversion
`scorpiox-debug.exe`	Debug utilities

`scorpiox-logger.exe`	Logging
`scorpiox-printlogs.exe`	Log viewer
`scorpiox-systemprompt.exe`	System prompt management
`scorpiox-rewind.exe`	Session rewind
`scorpiox-otp.exe`	OTP generation

Network & API Tools

Binary	Purpose
`scorpiox-fetch.exe`	URL content fetcher
`scorpiox-websearch.exe`	Web search tool
`scorpiox-screenshot.exe`	Screenshot capture
`scorpiox-renderimage.exe`	Image rendering
`scorpiox-email.exe`	SMTP email
`scorpiox-server-email.exe`	Email server
`scorpiox-dns.exe`	DNS server
`scorpiox-beam.exe`	LAN file transfer
`scorpiox-frp.exe`	FRP reverse proxy
`scorpiox-executecurl.exe`	cURL executor
`scorpiox-server.exe`	HTTP server
`scorpiox-server-fetchtoken.exe`	Token bridge
`scorpiox-server-http2tcp.exe`	HTTP-to-TCP relay
`scorpiox-hook.exe`	Webhook handler
`scorpiox-kql.exe`	KQL query tool

AI Provider Integrations

Binary	Purpose
`scorpiox-gemini.exe`	Gemini API proxy
`scorpiox-openai.exe`	OpenAI API proxy
`scorpiox-claudecode-fetchtoken.exe`	Claude Code token fetcher
`scorpiox-claudecode-refreshtoken.exe`	Claude Code token refresher
`scorpiox-claudecode-models.exe`	Claude Code model listing
`scorpiox-codex-fetchtoken.exe`	Codex token fetcher
`scorpiox-codex-refreshtoken.exe`	Codex token refresher
`scorpiox-gemini-fetchtoken.exe`	Gemini token fetcher

Agent & Development Tools

Binary	Purpose
`scorpiox-agent.exe`	AI agent framework
`scorpiox-tasks.exe`	Task management

`scorpiox-skills.exe`	Skill management
`scorpiox-planmode.exe`	Plan mode
`scorpiox-askuserquestion.exe`	User interaction
`scorpiox-runtest.exe`	Test runner
`scorpiox-mcp.exe`	MCP server
`scorpiox-host.exe`	Host management
`scorpiox-sdk.exe`	SDK tools
`scorpiox-search.exe`	Code search
`scorpiox-docs.exe`	Documentation

Infrastructure Tools

Binary	Purpose
`scorpiox-mirror-git.exe`	Git fleet mirror
`scorpiox-vault-git.exe`	Git vault
`scorpiox-usage.exe`	Usage telemetry (disabled by default)
`scorpiox-emit-session.exe`	Session telemetry (disabled by default)
`scorpiox-voice.exe`	Voice transcription
`scorpiox-transcript.exe`	Transcript viewer

2.2 Linux/macOS-Only Binaries (Not Deployed on Windows)

Binary	Purpose	Platform
`scorpiox-unshare`	Rootless container runtime (namespaces)	Linux
`scorpiox-vm`	KVM virtual machine runner	Linux
`scorpiox-init`	Container init process	Linux
`scorpiox-sshpass`	SSH password feeder	Linux
`scorpiox-traffic`	MITM traffic proxy	Linux
`scorpiox-podman`	Podman container wrapper	Linux
`scorpiox-cron`	Cron job manager	Linux
`scorpiox-whatsapp`	WhatsApp bridge	Linux
`scorpiox-thunderbolt4`	Thunderbolt4 test tool	Linux
`scorpiox-imessage`	iMessage integration	macOS

2.3 Windows-Filtered Findings

When scoped to Windows workstation deployment only, findings that affect exclusively Linux/macOS code paths are excluded:

Severity	All Platforms	Windows Only	Excluded
Critical	9	4	5 (iMessage injection ×2, WASM temp files ×3)
High	20	18	2 (traffic proxy, Linux server cache)
Medium	33	29	4 (unshare, container, Linux-only paths)

Low	29	29	0
Total	**212**	**80** + 121 info	—

---

3. Changes Since Last Audit

Previous Audit: Commit `28b0b1b` (2026-04-28)
Previous Total Findings: 274
Current Total Findings: 212
Net Change: −62 findings (22.6% reduction)

Severity	Previous	Current	Delta
Critical	7	9	+2
High	20	20	0
Medium	89	33	**−56**
Low	55	29	**−26**
Info	103	121	+18
Total	**274**	**212**	**−62**

Windows-Specific Delta:

Severity	Previous	Current	Delta
Critical	7	4	−3
High	20	18	−2
Medium	80	29	**−51**
Low	52	29	−23
Total	**159**	**80**	**−79**

Analysis: The codebase shows significant improvement since the previous audit, with a 22.6% overall reduction in findings and a 50% reduction in Windows-scoped findings. The medium-severity category saw the largest improvement (−56 findings), likely due to remediation of systemic patterns. The slight increase in critical findings (+2) is attributable to expanded agent coverage (install script audit now captures additional distribution security concerns). Informational findings increased due to more thorough endpoint cataloguing.

---

4. Supply Chain & Dependencies

Agent Risk Rating: LOW

4.1 Dependency Overview

The project has a minimal supply chain footprint with **zero runtime package managers** for the core build:

Component	Dependencies	Risk
Core C Application	0 runtime deps (vendored mbedTLS + yyjson)	LOW
Build System	CMake 3.16+, GCC/Clang, system libcurl, OpenSSL, zlib	LOW

Bridge (optional)	2 npm packages (`@whiskeysockets/baileys` , `qrcode-terminal`)	MEDIUM
Dockerfile (build-only)	`pip install requests` (unpinned)	LOW

4.2 Vendored Libraries

Library	Version	License	Lines	Assessment
mbedTLS	3.6.6 (LTS)	Apache-2.0 / GPL-2.0+	208,584	✅ Well-audited crypto library
yyjson	0.12.0	MIT	19,556	✅ Lightweight JSON parser

4.3 Findings

ID	Severity	Finding
SC-01	MEDIUM	No `package-lock.json` for bridge component — dependency versions not pinned
SC-02	LOW	Unpinned `pip install requests` in Dockerfile (build-time only)
SC-03	INFO	Zero pre-built binaries committed to repository
SC-04	INFO	No FetchContent, vcpkg, or Conan usage
SC-05	INFO	mbedTLS uses custom minimal config (`sx_mbedtls_config.h`)

---

5. Build System & Code Provenance

Agent Risk Rating: LOW

5.1 Security Hardening Flags

Category	Flags	Status
Warnings	`-Wall -Wextra`	✅ Enabled
Stack Protection	`-fstack-protector-strong`	✅ Enabled
Buffer Overflow	`-D_FORTIFY_SOURCE=2`	✅ Enabled (Release)
Control Flow	`-fcf-protection`	✅ GCC (when available)
RELRO	`-Wl,-z,relro,-z,now`	✅ Full RELRO
Non-exec Stack	`-Wl,-z,noexecstack`	✅ Enabled
Static Linking	`-static`	✅ Default (eliminates LD_PRELOAD attacks)

5.2 Findings

ID	Severity	Finding
BP-01	LOW	No reproducible build mechanism (no build hash verification)
BP-02	INFO	Security hardening flags are comprehensive
BP-03	INFO	Static linking eliminates shared library injection class

BP-04	INFO	No pre-built binaries in repository
BP-05	INFO	Build containers use official base images
BP-06	INFO	Version embedded at compile time via CMake

---

6. Network Endpoints

Agent Risk Rating: HIGH

6.1 Overview

78 unique hardcoded endpoints were identified across 38 source files. The software contacts well-defined API endpoints for its AI assistant functionality.

6.2 Authorized External Endpoints

Category	Endpoints
AI Providers	`api.anthropic.com`, `generativelanguage.googleapis.com`, OpenAI-compatible endpoints
Infrastructure	`code.scorpiox.net`, `dist.scorpiox.net`, `token.scorpiox.net`, `git.scorpiox.net`
Search	`search.mojeek.com`, `api.search.brave.com`
Voice	`whisper.scorpiox.net`

6.3 Findings

ID	Severity	Finding
NE-01	**CRITICAL**	Hardcoded Git PAT in release script — full repo read/write access
NE-02	HIGH	Plaintext HTTP for `token.scorpiox.net/codex` token endpoint
NE-03	HIGH	Plaintext HTTP for `token.scorpiox.net/gemini` token endpoint
NE-04	HIGH	Plaintext HTTP for localhost token proxy default
NE-05	MEDIUM	5 hardcoded private IP addresses expose internal topology
NE-06	MEDIUM	Hardcoded public Azure IP (`20.53.240.54`) as TCP_HOST
NE-07	MEDIUM	`0.0.0.0` bind addresses on server/DNS/beam tools
NE-08	LOW	Hardcoded public DNS resolvers (8.8.8.8, 1.1.1.1)
NE-09	LOW	Listen on 0.0.0.0 for LAN services

---

7. TLS/SSL Security

Agent Risk Rating: MEDIUM

7.1 TLS Posture

The software uses two TLS stacks: **libcurl** for HTTP/SMTP and **mbedTLS** (vendored) for FRP tunnels, mail relay, and IMAPS. Most connections enforce certificate verification and use TLS 1.2+.

7.2 Findings

ID	Severity	Finding
TLS-01	HIGH	Hardcoded plaintext HTTP defaults for token fetch endpoints (defense-in-depth gap)
TLS-02	MEDIUM	`sx_http.c` — no explicit SSL peer/host verification (relies on libcurl defaults)
TLS-03	MEDIUM	`scorpiox-pwsh.c` — no explicit SSL verification settings
TLS-04	MEDIUM	`scorpiox-claudecode-models.c` — no explicit SSL verification, no CA bundle
TLS-05	MEDIUM	Mail relay verification downgrade on specific certificate errors
TLS-06	LOW	Traffic proxy disables TLS verification (by design — MITM tool)
TLS-07	LOW	mbedTLS config allows TLS 1.2 minimum (acceptable)
TLS-08	LOW	FRP client uses mbedTLS custom CA path

---

8. Hardcoded Credentials

Agent Risk Rating: LOW

8.1 Assessment

The codebase uses a configuration-driven approach where sensitive values are loaded from config files or environment variables at runtime. Most credential fields in templates are **empty strings** — correct practice.

8.2 Findings

ID	Severity	Finding
HC-01	LOW	Placeholder `OPENAI_API_KEY=xxx` in config template (not a real key)
HC-02	INFO	Hardcoded public IP `20.53.240.54` as TCP_HOST default
HC-03	INFO	Internal IPs in config templates and release scripts
HC-04	INFO	Service URLs hardcoded in C source (expected for service discovery)
HC-05	INFO	SSH config defaults with usernames/ports
HC-06	INFO	DKIM RSA test key structure in email module

HC-07	INFO	JWT issuer/audience URLs in server component

---

9. File I/O & Data Handling

Agent Risk Rating: HIGH

9.1 Key Findings

ID	Severity	Finding	Windows Impact
FIO-01	**CRITICAL**	Predictable temp file `/tmp/.sx_pipe_data` — symlink attack	✗ WASM/Linux only
FIO-02	**CRITICAL**	Predictable temp file `/tmp/.sx-unshare-list.html` — symlink attack	✗ Linux only
FIO-03	**CRITICAL**	Predictable temp file `/tmp/.sx_stdin_data` — symlink attack	✗ WASM/Linux only
FIO-04	HIGH	OTP secret passed via command line (visible in process listing)	✓ Windows
FIO-05	HIGH	Full OAuth tokens printed to stdout in verbose mode	✓ Windows
FIO-06	HIGH	API keys visible in `/proc/*/environ` on Linux	✗ Linux only
FIO-07	HIGH	Mail accounts file has no permission restrictions	✓ Windows
FIO-08	HIGH	Server git clone cache in world-readable `/tmp`	✗ Linux only
FIO-09	MEDIUM	40+ missing NULL checks after `fopen()`	✓ Windows
FIO-10	MEDIUM	WASM `mktemp` uses `fopen` without `O_EXCL` (TOCTOU race)	✗ WASM only
FIO-11	MEDIUM	DNS stats/PID files in predictable `/tmp` locations	✓ Windows (lesser risk)
FIO-12	MEDIUM	World-writable device nodes in container	✗ Linux only
FIO-13	MEDIUM	Container rootfs stored in world-readable location	✗ Linux only
FIO-14	MEDIUM	Voice WAV files stored in predictable `/tmp` path	✓ Windows

10. Buffer Safety

Agent Risk Rating: LOW

10.1 String Function Usage

Category	Count
`snprintf` (safe)	2,213
`strncpy` (safe)	592
`strncat` (safe)	20
`calloc` (safe)	99
`strcpy` (unsafe)	**0**
`sprintf` (unsafe)	**0**
`gets` (unsafe)	**0**
`strcat` (unsafe)	**11**
Safe:Unsafe Ratio	**266:1**

10.2 Findings

ID	Severity	Finding	Windows Impact
BUF-01	MEDIUM	`strcat` in unbounded loop with `size_t` underflow risk (Windows `system()` path)	✅ Windows only
BUF-02	LOW	Fixed-size stack buffers in argument processing (8KB sufficient for typical use)	✅ Windows
BUF-03	INFO	Vendor yyjson has 1 conditional `sprintf` with `#if` preferring safe variants	ℹ N/A

11. Memory Safety

Agent Risk Rating: MEDIUM

11.1 Allocation Statistics

Metric	Count
`malloc` / `calloc` calls	389
`realloc` calls	110
`strdup` / `strndup` calls	415
`free()` calls	1,341

11.2 Findings

ID	Severity	Finding
MEM-01	HIGH	Memory leak on `realloc` failure in `sx_agent.c` — original buffer not freed
MEM-02	MEDIUM	168 unchecked `strdup()` return values across 39 files — NULL propagation risk
MEM-03	LOW	Redundant NULL check pattern after `realloc` (cosmetic, not a bug)
MEM-04	LOW	`realloc` old-pointer safety — correctly handled in 107/110 call sites

---

12. Command Injection

Agent Risk Rating: HIGH

12.1 Attack Surface

- ~55 `system()` calls, ~30 `popen()` calls, ~45 `exec*()` calls across ~20 source files
- Unix code paths use `fork()+execvp()` (safe) for most operations
- Windows code paths use `system()` with command string construction

12.2 Findings

ID	Severity	Finding	Windows Impact
CJ-01	**CRITICAL**	Direct shell-out of user input via `!` command (`system(cmd)`)	✅ By design
CJ-02	**CRITICAL**	iMessage AppleScript injection via message/recipient	❌ macOS only
CJ-03	**CRITICAL**	iMessage attachment path injection	❌ macOS only
CJ-04	HIGH	Windows `system()` command construction with `strcat` (buffer+injection)	✅ Windows
CJ-05	HIGH	`scorpiox-podman.c` — `snprintf` → `system()` with container names	❌ Linux only
CJ-06	HIGH	`scorpiox-sdk.c` — `system()` with workspace path	✅ Windows
CJ-07	HIGH	`scorpiox-tmux.c` — `system()` with session names	✅ Windows
CJ-08	MEDIUM	`is_safe_shell_arg()` validation bypass via crafted strings	✅ Windows
CJ-09	MEDIUM	Environment variable injection via config values into `system()`	✅ Windows

CJ-10	MEDIUM	DNS server `system()` for stats display	✅ Windows
CJ-11	MEDIUM	Voice tool `system()` for audio playback	✅ Windows
CJ-12	MEDIUM	Container-only paths use `system()`	❌ Linux only

Note: CJ-01 (the `!` shell-out) is an intentional feature of the interactive CLI — the user explicitly requests shell command execution. The risk is indirect prompt injection where an AI model might suggest malicious commands.

---

13. Privilege & Access Control

Agent Risk Rating: MEDIUM

13.1 Privilege Model

- No setuid/setgid binaries
- No root privileges required for core binary
- All operations run under user context
- Container runtime (`scorpiox-unshare`) uses rootless Linux user namespaces

13.2 Findings

ID	Severity	Finding	Windows Impact
PA-01	HIGH	`system()` with AI-controlled command strings in `sx.c`	✅ By design
PA-02	HIGH	`scorpiox-bash.c` arbitrary command execution via `fork+execl`	✅ By design
PA-03	MEDIUM	DNS server defaults to port 53 (requires elevated privileges)	✅ Windows
PA-04	MEDIUM	VM runner requires `/dev/kvm` access	❌ Linux only
PA-05	MEDIUM	`system()` used in 20+ locations without input sanitization	✅ Windows
PA-06	MEDIUM	Container runtime skips `CLONE_NEWUSER` when running as root	❌ Linux only
PA-07	MEDIUM	Predictable temp path in unshare	❌ Linux only
PA-08	LOW	`scorpiox-sshpass` feeds passwords via PTY (by design)	❌ Linux only
PA-09	LOW	Extensive `pipe()` usage (~40 sites) — all properly error-checked	✅ Informational
		`popen()` used in ~51	

PA-10	LOW	locations for output capture	✅ Windows

---

14. Windows Deployment Analysis

Agent Risk Rating: LOW

14.1 Windows Build Configuration

- **Compiler:** MinGW GCC (cross-compile or native)
- **Linking:** Fully static with `.a` libraries (libcurl, OpenSSL, zlib)
- **Security Flags:** Stack protector, FORTIFY_SOURCE, full warnings
- **No DLL dependencies** — eliminates DLL hijacking risk

14.2 Windows Behavior Assessment

Category	Assessment
Network Contacts	Well-defined API endpoints only; no undisclosed connections
File System Access	User-controlled paths in `.scorpiox/` directories
Process Spawning	Shells and scripts with user permissions only
Windows APIs	Standard Winsock, GDI, WinInet — appropriate usage
Backdoors/Rootkits	None identified
Data Exfiltration	None — all telemetry disabled by default
Auto-Update	WSL component checks `dist.scorpiox.net` for binary updates

14.3 Findings

All 12 findings are **informational**, documenting expected behaviors:

ID	Finding
WD-01 through WD-12	Documented binary purposes, network endpoints, file I/O patterns, and process spawning behavior — all consistent with stated application purpose

---

15. Telemetry and Tracking

Agent Risk Rating: LOW

15.1 Telemetry Status

Verdict: No tracking by default.

All telemetry features are **disabled by default** across all configuration layers (compiled-in defaults, runtime config, and config templates). No network calls related to telemetry occur unless explicitly enabled by the user.

15.2 Telemetry Features

Feature	Default	Binary	Data Sent	Conversation Content
`USAGE_TRACKING`	**OFF** (`0`)	`scorpiox-`	Token counts +	✗ No

`USAGE_TRACKING`	**OFF** (`0`)	`usage`	machine metadata	✗ No
`EMIT_SESSION_TRACKING`	**OFF** (`0`)	`scorpiox-emit-session`	Full message content + metadata	⚠ Yes (when enabled)

15.3 Findings

ID	Severity	Finding
TEL-01	LOW	Stale documentation claims `USAGE_TRACKING` defaults to `1` — actual code default is `0`
TEL-02	LOW	Session telemetry sends full conversation content when explicitly enabled
TEL-03	INFO	No auto-update mechanism phones home at startup
TEL-04	INFO	No analytics, crash reporting, or fingerprinting at rest
TEL-05	INFO	Machine metadata (hostname, username, OS) sent with usage data when enabled
TEL-06	INFO	Both telemetry features use separate binaries (fire-and-forget, non-blocking)
TEL-07	INFO	WhatsApp bridge update is on-demand only (not automatic)

---

16. Install Script & Distribution Security

Agent Risk Rating: CRITICAL

16.1 Distribution Model

The software uses a `curl|bash` install pattern (`curl -fsSL "get.scorpiox.net?platform=linux" | bash`) with archives served from `dist.scorpiox.net` .

16.2 Findings

ID	Severity	Finding
INS-01	**CRITICAL**	`curl|bash` install with unauthenticated script endpoint — script source not in repository
INS-02	**CRITICAL**	No cryptographic code signing of released binaries or archives (no GPG, cosign, sigstore)
INS-03	HIGH	SHA256 verification degrades gracefully — downloads proceed without integrity check if `.sha256` unavailable
INS-04	HIGH	Install script endpoint (`get.scorpiox.net`) not auditable — not in repository
INS-05	HIGH	Windows auto-update applies binaries on startup without explicit user consent

INS-06	HIGH	Distribution via SMB to file server — no pipeline integrity chain
INS-07	MEDIUM	Documented `curl -k` usage (disables TLS verification) in operational notes
INS-08	MEDIUM	Container install extracts to `/usr/local/bin` as root without verification
INS-09	MEDIUM	`index.txt` branch resolution via unauthenticated HTTP fetch
INS-10	MEDIUM	No HSTS or security header enforcement visible
INS-11	MEDIUM	Archive extraction without path traversal validation
INS-12	LOW	Release scripts use environment variables for passwords (good) but hardcode IPs
INS-13	LOW	No SBOM (Software Bill of Materials) generated in release pipeline
INS-14	LOW	Version check uses simple string comparison

---

17. Consolidated Risk Matrix

Area	Finding	Severity	Status	Platform	Recommendation
Distribution	No code signing of binaries	**CRITICAL**	Open	All	Implement GPG/cosign signing for all release artifacts
Distribution	`curl|bash` with unauditable script	**CRITICAL**	Open	Linux	Include install script in repository; add signature verification
Network	Hardcoded Git PAT in release script	**CRITICAL**	Open	Build Infra	Rotate PAT immediately; use credential helpers
Command Injection	`!` shell-out of user input	**CRITICAL**	By Design	All	Ensure non-interactive paths cannot trigger; add confirmation
File I/O	Predictable temp files (WASM)	**CRITICAL**	Open	Linux/WASM	Replace with `mkstemp`-based unique paths
Command Injection	iMessage AppleScript injection	**CRITICAL**	Open	macOS	Replace `system()` with `fork()+execvp()`
Network	Plaintext HTTP for token endpoints	HIGH	Open	All	Change `DEFAULT_REMOTE_URL` to `https://`
TLS	Missing explicit SSL verification	HIGH	Open	All	Add explicit `CURLOPT_SSL_VERIFYPEER` settings
Distribution	SHA256 verification optional/degraded	HIGH	Open	All	Fail closed when checksums unavailable
	Auto-update				Add explicit user

Distribution	without user consent	HIGH	Open	Windows	confirmation for binary updates
File I/O	OTP secret in command line	HIGH	Open	Windows	Pass via environment variable or stdin
File I/O	OAuth tokens in verbose output	HIGH	Open	All	Redact tokens in debug output
Memory	Leak on realloc failure in agent	HIGH	Open	All	Free original buffer before returning NULL
Memory	168 unchecked `strdup()` returns	MEDIUM	Open	All	Add NULL checks or create `sx_strdup()` wrapper
Buffer	`strcat` with `size_t` underflow	MEDIUM	Open	Windows	Replace with bounded string operations
TLS	Implicit SSL verification defaults	MEDIUM	Open	All	Make verification explicit in all HTTP clients
File I/O	40+ missing fopen NULL checks	MEDIUM	Open	All	Add NULL checks and error handling
Privilege	DNS server on port 53 (elevated)	MEDIUM	Open	All	Default to unprivileged port; document requirement
Supply Chain	No lockfile for bridge npm deps	MEDIUM	Open	Linux	Add `package-lock.json`
Telemetry	Stale documentation for tracking default	LOW	Open	All	Update help text to match code default
Credentials	Placeholder API key in template	LOW	Open	All	Clear placeholder value

---

18. Conclusion & Recommendations

18.1 Overall Assessment

ScorpioX Code demonstrates **above-average security posture** for a C codebase of this size (380K+ lines). The development team has made conscious security decisions including:

- **Zero use** of `strcpy`, `sprintf`, `gets` — replaced entirely with bounded alternatives
- **Full static linking** eliminating dynamic library attacks
- **Comprehensive compiler hardening** (stack protector, FORTIFY_SOURCE, full RELRO, CFI)
- **Telemetry disabled by default** with explicit opt-in required
- **No pre-built binaries** in repository
- **Configuration-driven secrets** with empty defaults

18.2 Priority Recommendations

P0 — Immediate (Critical): 1. **Rotate the hardcoded Git PAT** in release scripts and move to credential helpers 2. **Implement code signing** for all release binaries (GPG or cosign) 3. **Change token endpoint defaults** from `http://` to `https://`

P1 — Short Term (High): 4. **Add explicit SSL verification** (`CURLOPT_SSL_VERIFYPEER=1L`,

`CURLOPT_SSL_VERIFYHOST=2L`) in all HTTP client code 5. **Fail closed** on missing SHA256 checksums during install/update 6. **Require user confirmation** for Windows auto-updates 7. **Fix realloc memory leak** in `sx_agent.c` 8. **Pass OTP secrets via stdin/environment** instead of command line

P2 — Medium Term (Medium): 9. **Create `sx_strdup()` wrapper** that aborts on OOM to address 168 unchecked sites 10. **Replace `strcat` with bounded operations** in Windows command builders 11. **Add NULL checks** after all `fopen()` calls 12. **Make SSL verification explicit** rather than relying on libcurl defaults 13. **Replace predictable temp files** with `mkstemp` in WASM/Linux paths

P3 — Long Term (Low): 14. Add `package-lock.json` for bridge component 15. Generate SBOM in release pipeline 16. Update stale documentation for telemetry defaults 17. Move infrastructure IPs to environment variables in release scripts

18.3 Windows Deployment Recommendation

For Windows workstation deployment, ScorpioX Code is assessed as LOW-MEDIUM risk. The most severe findings (iMessage injection, WASM temp file attacks, Linux container vulnerabilities) do not apply to the Windows platform. The primary Windows-relevant concerns are:

- Distribution integrity (no code signing) — **mitigated if distributed via internal channels with verified hashes**
- Plaintext HTTP token endpoint defaults — **mitigated by runtime config that correctly uses HTTPS**
- Buffer overflow in Windows command builder — **exploitable only with extremely long argument lists**

With the P0 recommendations addressed (PAT rotation, code signing, HTTPS defaults), the Windows deployment risk would be reduced to **LOW**.

---

19. Appendix: Audit Methodology

19.1 Agents Deployed

#	Agent	Scope
1	supply-chain	Package managers, vendored libraries, dependency graph
2	build-provenance	CMake config, compiler flags, linking strategy, reproducibility
3	network-endpoints	Hardcoded URLs, IPs, domains, ports across all source files
4	tls-security	SSL/TLS configuration, certificate verification, cipher suites
5	credential-hardcode	API keys, passwords, tokens, secrets in source and config
6	file-io	File operations, temp files, permissions, data at rest
7	buffer-safety	Buffer overflows, format strings, integer overflows, unsafe functions
8	memory-safety	malloc/free patterns, use-after-free, double-free, leaks, strdup
9	command-injection	system(), popen(), exec*() with user-controlled input

10	privilege-access	SUID/SGID, root requirements, sandboxing, IPC, process spawning
11	windows-deployment	Windows binary inventory, behavior analysis, API usage
12	telemetry-tracking	Data collection, phone-home behavior, opt-in/opt-out analysis
13	install-script	Distribution mechanism, install scripts, update process, signing

19.2 Analysis Approach

- **Static analysis** of all 149 non-vendor C source files (151,970 lines)
- **Configuration review** of CMakeLists.txt, Dockerfiles, release scripts
- **Pattern matching** for known-dangerous function calls and patterns
- **Data flow analysis** for user input reaching security-sensitive sinks
- **Cross-reference** between vendor library versions and known CVE databases
- **Platform-specific filtering** for Windows deployment scope

19.3 Limitations

- This audit covers source code analysis only; no dynamic testing was performed
- The install script hosted at `get.scorpiox.net` is not included in the repository and could not be audited
- Runtime behavior may differ from static analysis findings due to configuration overrides
- Vendor library internals (mbedTLS, yyjson) were not deeply audited — version checks only

19.4 Classification

This report is classified as **CONFIDENTIAL — For Corporate Review Only**. Distribution should be limited to security, engineering, and IT leadership teams involved in the software evaluation and deployment decision.

Report generated by ScorpioX Code Security Audit Pipeline — 13-agent automated analysis
Audit commit: `40e8525bd1b39a2512668945e5fbdddaf1101ccc`