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:** 4d61a6ad4bd4c6e535dd7f57d3eff860eaf61168 **Classification:** CONFIDENTIAL — For Corporate Review Only

---

1. Executive Summary

This report presents a comprehensive security audit of ScorpioX Code, an AI-powered development tool and CLI platform written in pure C with vendored dependencies. The audit was conducted across **11 specialized security domains** by independent automated agents analyzing the same codebase at commit `4d61a6a`.

Key Metrics

Metric	Value
Total Source Lines	398,610
First-Party Code	170,860 lines (42.9%)
Vendored Code	227,750 lines (57.1%)
Total Findings	395
Critical	10
High	25
Medium	70
Low	123
Informational	167
Overall Risk Rating	**HIGH**

Assessment Overview

The software demonstrates **strong fundamentals** in several areas: zero use of unsafe C string functions (`sprintf` , `strcpy` , `strcat`), fully static linking, no setuid binaries, proper namespace isolation in container runtimes, and no external package manager dependencies for the core build. Two well-known vendored libraries (Mbed TLS 3.6.3, yyjson 0.12.0) are properly maintained and licensed.

However, the audit identified **significant security gaps** in three critical areas:

1. **TLS/Certificate Verification (CRITICAL):** SSL certificate verification is disabled on all token-fetching endpoints. Authentication tokens are transmitted over plaintext HTTP. The traffic proxy globally disables TLS verification for child processes.
2. **Command Injection (HIGH):** Despite a recent commit specifically fixing command injection

(the HEAD commit), numerous residual `system()` / `popen()` calls remain where network-sourced or user-controlled data flows into shell command strings without adequate escaping — particularly in messaging integrations.

3. **Memory Safety (HIGH):** Systemic missing-NULL-check patterns across 495 allocation calls, several use-after-free vulnerabilities in network-facing server components, and 35 unsafe `realloc()` patterns that leak memory on failure.

The **Windows deployment surface** is significantly narrower: 10 Linux/macOS-only binaries (including the container runtimes and messaging bridges where the most critical command injection and privilege findings reside) are excluded from Windows builds.

---

2. Deployment Scope — Windows Workstation vs. Server-Side

2.1 Primary Deployment: Windows Workstation

ScorpioX Code is deployed as a **statically-linked Windows application** distributed as a ZIP archive containing executables and bundled GNU/BusyBox Unix tools. The build uses MinGW (WinLibs POSIX UCRT) with static linking against all libraries.

2.2 Windows Binaries (54 total)

The following executables are compiled for and deployed on Windows:

Category	Binaries
Core TUI & Shell	`sx.exe`, `scorpiox-bash.exe`, `scorpiox-tmux.exe`, `scorpiox-multiplexer.exe`, `scorpiox-vi.exe`, `scorpiox-config.exe`
AI Provider Proxies	`scorpiox-gemini.exe`, `scorpiox-openai.exe`, `scorpiox-beam.exe`, `scorpiox-host.exe`
Token Management	`scorpiox-claudecode-fetchtoken.exe`, `scorpiox-claudecode-refreshtoken.exe`, `scorpiox-codex-fetchtoken.exe`, `scorpiox-codex-refreshtoken.exe`, `scorpiox-gemini-fetchtoken.exe`, `scorpiox-server-fetchtoken.exe`, `scorpiox-claudecode-models.exe`
Tools & Utilities	`scorpiox-websearch.exe`, `scorpiox-fetch.exe`, `scorpiox-renderimage.exe`, `scorpiox-screenshot.exe`, `scorpiox-executecurl.exe`, `scorpiox-kql.exe`
Agent & SDK	`scorpiox-agent.exe`, `scorpiox-sdk.exe`, `scorpiox-tasks.exe`, `scorpiox-skills.exe`, `scorpiox-planmode.exe`, `scorpiox-askuserquestion.exe`, `scorpiox-mcp.exe`, `scorpiox-hook.exe`, `scorpiox-runtest.exe`
Session & Logging	`scorpiox-conv.exe`, `scorpiox-rewind.exe`, `scorpiox-debug.exe`, `scorpiox-logger.exe`, `scorpiox-printlogs.exe`, `scorpiox-transcript.exe`, `scorpiox-systemprompt.exe`
Infrastructure	`scorpiox-server.exe`, `scorpiox-server-http2tcp.exe`, `scorpiox-server-email.exe`, `scorpiox-email.exe`, `scorpiox-dns.exe`, `scorpiox-frp.exe`
Telemetry	`scorpiox-usage.exe`, `scorpiox-emit-session.exe`
Windows-Only	`scorpiox-wsl.exe` (WSL2 runtime), `scorpiox-`

	`busybox.exe` (Unix tools manager)
Other	`scorpiox-voice.exe`, `scorpiox-otp.exe`, `scorpiox-search.exe`, `scorpiox-docs.exe`, `scorpiox-vault-git.exe`, `scorpiox-mirror-git.exe`, `scorpiox-pwsh.exe`

2.3 Linux/macOS-Only Binaries — NOT Deployed on Windows

Binary	Platform	Reason
`scorpiox-unshare`	Linux	Linux user namespaces, `sys/mount.h`, `sys/syscall.h`
`scorpiox-vm`	Linux	Linux KVM ioctls
`scorpiox-init`	Linux	Container PID 1, Linux-only syscalls
`scorpiox-sshpass`	Unix	PTY (`forkpty`), Unix-only
`scorpiox-traffic`	Unix	Network traffic interception, Unix sockets
`scorpiox-podman`	Unix	Podman container execution
`scorpiox-cron`	Unix	Crontab wrapper
`scorpiox-whatsapp`	Unix	Uses `fork` / `pipe` / `select`
`scorpiox-thunderbolt4`	macOS	macOS framework APIs
`scorpiox-imessage`	macOS	macOS iMessage integration

2.4 Windows-Filtered Findings

When scoped to **Windows workstation deployment only** (excluding Linux/macOS-only binaries), the finding counts are reduced:

Severity	All Platforms	Windows Only	Excluded
Critical	10	7	3 (traffic proxy TLS bypass, WhatsApp/iMessage injection)
High	25	20	5 (WhatsApp/iMessage/Podman/unshare injection, unshare privilege)
Medium	70	62	8 (WhatsApp injection, namespace isolation gaps)
Low	123	119	4 (Linux-specific)
Total	**395**	**208**	**187**

---

3. Changes Since Last Audit

The most recent previous audit was conducted on the same date against commit `f7f14b4`.

Metric	Previous (f7f14b4)	Current (4d61a6a)	Delta
Overall Risk	MEDIUM	HIGH	⬆ Increased
Audit Agents	5 (general)	11 (specialized)	+6
Total Findings	40	395	+355
Critical	5	10	+5
High	12	25	+13

Medium	14	70	+56
Low	6	123	+117
Info	3	167	+164
Lines of Code	369,495	398,610	+29,115

Interpretation: The significant increase in findings is primarily attributable to the **expanded audit scope** — from 5 general-purpose agents to 11 specialized agents with deeper analysis (particularly the memory safety agent which contributed 230 findings alone, including 92 LOW-severity missing-NULL-checks and 93 INFO-severity leak candidates). The HEAD commit (`security: fix command injection — replace system()/popen() with fork+execvp`) demonstrates active remediation of previously identified command injection vectors.

---

4. Supply Chain & Dependencies

Agent Risk Rating: LOW

Key Findings

The core build has **zero external package manager dependencies**. All third-party code is vendored in-tree:

Library	Version	Lines	License	Assessment
Mbed TLS	3.6.3 (LTS)	207,697	Apache-2.0 / GPL-2.0+	✅ Well-established, ARM-maintained
yyjson	0.12.0	19,556	MIT	✅ Widely-used JSON parser

System dependencies (linked statically): libcurl, OpenSSL, zlib, pthreads, nghttp2/3, brotli, zstd, c-ares, libpsl, libidn2, libunistring.

#	Severity	Finding
1	MEDIUM	Two pre-built ELF binaries (`bridge/scorpiox-ws2tcp` , `bridge/scorpiox-ws2tcp-arm64`) committed to repository without reproducible build provenance
2	LOW	npm dependency tree (~104 transitive packages) in WhatsApp bridge component via community-maintained `@whiskeysockets/baileys`
3	LOW	MSYS2 download script (`vendor/gnuwin64/download.ps1`) lacks package hash verification
4	INFO	Vendored code represents 57.1% of total codebase (dominated by Mbed TLS)
5	INFO	All vendored libraries are well-known, properly licensed, at recent versions

---

5. Build System & Code Provenance

Agent Risk Rating: LOW-MEDIUM

Build System

- **Primary:** CMake 3.16+ with static linking (`SX_STATIC_LINK=ON`)
- **Windows:** MinGW (WinLibs POSIX UCRT) via Ninja
- **66 executable targets** defined in `scorpiox/CMakeLists.txt`

Security Hardening Gaps

The following compiler hardening flags are **absent** from all build configurations:

Flag	Purpose	Status
`-fstack-protector-strong`	Stack buffer overflow detection	✗ Missing
`-D_FORTIFY_SOURCE=2`	Runtime buffer overflow checks	✗ Missing
`-fPIE` / `-pie`	ASLR (position-independent executables)	✗ Missing
`-Wl,-z,relro,-z,now`	Full RELRO (GOT protection)	✗ Missing
`-fstack-clash-protection`	Stack clash prevention	✗ Missing
`-fcf-protection`	Control-flow integrity	✗ Missing

> **Note:** Static linking provides inherent protection against shared library attacks, but the absence of stack protectors and FORTIFY_SOURCE remains a gap.

Code Provenance

#	Severity	Finding
1	MEDIUM	Missing compiler security hardening flags (stack protector, FORTIFY_SOURCE, PIE, RELRO)
2	MEDIUM	Build-time Python script fetches model IDs from external API (currently frozen/disabled)
3	LOW	Git history shows 99%+ single-organization commits
4	LOW	`strip` applied to release binaries (good practice, minor debug impact)
5	LOW	No code signing for released binaries

---

6. Network Endpoints

Agent Risk Rating: HIGH

Endpoint Inventory

The codebase contains **53 unique hardcoded endpoints** in project code:

Category	Count	Protocol
Token/Auth endpoints	9	HTTP (plaintext) ⚠
Distribution/Update URLs	12	HTTPS ✓
API service endpoints	8	HTTPS ✓
Internal IP addresses	5	Various ⚠

Localhost bindings	7	TCP
Third-party APIs	12	HTTPS ✅

Critical Network Findings

#	Severity	Finding
1	HIGH	9 token-fetching endpoints use plaintext `http://` — authentication tokens transmitted in cleartext
2	MEDIUM	5 hardcoded private IP addresses embedded in source/config (information disclosure of internal topology)
3	LOW	7 hardcoded default ports and localhost bindings
4	INFO	38 HTTPS URLs properly encrypted for distribution, API, and third-party services

---

7. TLS/SSL Security

Agent Risk Rating: CRITICAL

This domain represents the most severe findings in the audit.

Critical Findings

ID	Finding	CVSS	Impact
C01	SSL certificate verification disabled (`CURLOPT_SSL_VERIFYPEER=0`, `CURLOPT_SSL_VERIFYHOST=0`) on all token-fetch endpoints (Gemini, Codex, Claude Code)	9.1	MITM can intercept OAuth tokens
C02	Default token endpoints use plaintext HTTP (`http://token.scorpiox.net/...`)	9.1	Cleartext credential transmission
C03	Traffic proxy globally disables TLS verification via environment variables (`NODE_TLS_REJECT_UNAUTHORIZED=0` , `CURLOPT_SSL_VERIFYPEER=0`)	8.1	All child processes vulnerable to MITM (Linux only)

Additional TLS Findings

#	Severity	Finding
1	HIGH	mbedTLS client connections configured without certificate verification (`MBEDTLS_SSL_VERIFY_NONE`)
2	HIGH	SMTP client (`sxmail_tls.c`) disables certificate verification for outbound mail relay
3	MEDIUM	No minimum TLS version enforcement via `CURLOPT_SSLVERSION`
		Router URL (`scorpiox.net:5176`) uses

4	MEDIUM	plaintext HTTP
5	MEDIUM	SMTP server supports STARTTLS but also allows plaintext fallback
6	LOW	No certificate pinning for high-value endpoints
7	INFO	Refresh token endpoints (`scorpiox-codex-refreshtoken` , `scorpiox-claudecode-refreshtoken`) correctly enable verification — inconsistency with fetch endpoints
8	INFO	mbedTLS server-mode `VERIFY_NONE` is acceptable (no mutual TLS requirement)

---

8. Hardcoded Credentials

Agent Risk Rating: MEDIUM

No active credentials (real API keys, production passwords, or live tokens) were found hardcoded in the codebase. Findings relate to placeholder values and internal infrastructure details.

#	Severity	Finding
1	MEDIUM	Placeholder OpenAI API key `"xxx"` hardcoded as default — triggers authentication attempts with garbage value
2	MEDIUM	Default SSH username `"root"` hardcoded for Claude Code token fetching
3	MEDIUM	Internal SSH host IP (`192.168.1.6`) in distributed config file with `SSH_USER=root` , `SSH_PORT=22223`
4	LOW	Internal IP `192.168.1.3` for TMUX remote host in embedded config
5	LOW	Internal IP addresses for OTP, voice, and server endpoints in config
6	LOW	Default SSH port `22223` and SSH key path in config
7	LOW	SSH password placeholder `sshpassword` in config
8	LOW	Codex SSH credentials pattern similar to Claude Code
9-12	INFO	API key fields empty by default (good practice); config masking function exists; no `.env` files committed

---

9. File I/O & Data Handling

Agent Risk Rating: MEDIUM

#	Severity	Finding
1	HIGH	Environment variables logged without sensitive value filtering in agent/server

		modules — API keys may appear in plaintext logs
2	HIGH	Predictable temp file names in `/tmp` (e.g., `/tmp/sx_voice.wav`, `/tmp/sx_emit_*.txt`) using `snprintf` instead of `mkstemp` — symlink attacks possible
3	MEDIUM	Insecure temp file creation via `fopen()` without `O_EXCL` in multiple modules
4	MEDIUM	Inconsistent directory permissions — some paths use `0700`, others use `0755` or default umask
5	MEDIUM	Missing `O_CLOEXEC` on file descriptors — potential leakage to child processes
6	MEDIUM	Configuration values (including sensitive ones) written to disk in plaintext
7	LOW	Missing `umask()` before file creation in several modules
8	LOW	`/tmp` files not cleaned on crash/signal
9	LOW	TOCTOU race conditions in temp file operations
10-12	INFO	Auth tokens properly redacted in traffic logs (positive); `sx_config_is_sensitive()` used for config logging (positive); session directories use `0700` (positive)

---

10. Buffer Safety

Agent Risk Rating: LOW

This is a **strong area** for the codebase.

Safe Function Usage (Project Code)

Function	Count	Safety
`snprintf`	2,248	✅ Bounded
`strncpy`	591	✅ Bounded
`strncat`	14	✅ Bounded
`sprintf`	0	✅ Not used
`strcpy` (bare)	0	✅ Not used
`strcat` (bare)	0	✅ Not used
`gets`	0	✅ Not used

Safe-to-unsafe ratio: ∞ — No unsafe string functions in project code.

#	Severity	Finding
1	MEDIUM	Signed `int` position accumulators in `snprintf()` loops can silently overflow, causing truncated command strings — potential for partial command execution

2	LOW	Large stack buffers (4–16 KB) with accumulation loops lack explicit truncation detection
3	LOW	`strncpy` usage does not always ensure NUL termination on exact-fit copies
4-8	INFO	Vendor code (yyjson) has single conditional `sprintf` fallback for pre-C99 compilers only; all mbedTLS format calls use `snprintf` ; no format string vulnerabilities found

---

11. Memory Safety

Agent Risk Rating: HIGH

The memory safety audit analyzed **495 allocation calls** and **1,309 `free()` calls** across 150 non-vendor source files. This domain produced the highest finding count.

Summary

Category	Count	Severity
Use-after-free in network/server paths	3	Critical
Use-after-free in agent/protocol handlers	7	High
Unsafe `realloc` (old pointer lost on failure)	35	Medium
Missing NULL check after `malloc` / `calloc` / `realloc`	92	Low
Early-return memory leak candidates	93	Info

Critical Memory Findings

ID	Finding	Location
CRIT-03	Multiple unchecked `malloc` calls in HTTP server request handler — NULL dereference crash causes denial of service	`scorpiox-server.c:2409, 2806`

> **Note:** Two initially-reported critical use-after-free findings (CRIT-01 in `sxmail_queue.c` and CRIT-02 in `ws2tcp.c`) were **verified as false positives** upon detailed analysis — control flow prevents the freed pointers from being used.

Systemic Patterns

- **Unsafe realloc (35 instances):** Pattern `ptr = realloc(ptr, new_size)` loses the old pointer on failure, causing memory leaks. Affects dynamic buffer growth in JSON parsing, HTTP response accumulation, and agent output collection.
- **Missing NULL checks (92 instances):** Most `malloc` / `calloc` calls proceed without verifying the allocation succeeded. On memory pressure, this results in NULL pointer dereference crashes.

---

12. Command Injection

Agent Risk Rating: HIGH

The HEAD commit (`4d61a6a: security: fix command injection — replace system()/popen() with fork+execvp`) demonstrates active remediation. However, residual vulnerabilities remain.

Critical Command Injection Findings

ID	Finding	CVSS	Network?	Windows?
C1	Hook data JSON injected into `popen()` / `system()` — single quotes not escaped	9.8	Yes (webhook events)	Yes
C2	WhatsApp message content injected into `system()` — display name, emoji, message ID	9.8	Yes (any WhatsApp sender)	No
C3	iMessage sender data injected into `system()` — phone number in folder path	8.8	Yes (any iMessage sender)	No
C4	`is_safe_shell_arg()` does not block single quotes — all callers using single-quote delimited `system()` calls are bypassable	9.0	Various	Yes

High Severity

ID	Finding	Network?	Windows?
H1	WSL command construction with user-controlled project names into `system()`	No	Yes
H2	Server config values interpolated into `system()` for agent spawning	Config	Yes
H3	iMessage attachment path injection	Yes	No
H4	WhatsApp OGG conversion with unsanitized paths	No	No
H5	Podman image/container names into `system()`	No	No
H6	Unshare overlay paths into `system()`	No	No
H7	Server OTP generation with query parameters (partially fixed)	Yes	Yes
H8	SSH proxy hostname injection in TMUX	Config	Yes

Recommendations

1. **Replace all remaining `system()` calls with `fork()` + `execvp()`** — the codebase already demonstrates this pattern correctly in `scorpiox-websearch.c` and the editor launch.

2. **Fix `is_safe_shell_arg()` to block single quotes** — add `'` to the rejected character list.
3. **Implement proper shell escaping** for any remaining `system()` / `popen()` usage.

---

13. Privilege & Access Control

Agent Risk Rating: MEDIUM

Positive Findings

- **No setuid/setgid binaries** shipped
- Container runtime (`scorpiox-unshare`) targets **rootless execution** via user namespaces
- Proper `pivot_root` with old root cleanup
- Environment cleared (`clearenv()`) and rebuilt with minimal set in containers
- cgroup v2 memory limits enforced
- slirp4netns with `--disable-host-loopback` for network isolation

Findings

#	Severity	Finding
1	HIGH	WSL2 runtime executes all container operations as root (`wsl.exe -d <distro> -u root`)
2	HIGH	Shell command strings built via `snprintf()` and passed to `system()` in WSL module
3	HIGH	Container runtime retains root-capable code paths when `getuid() == 0` (skips user namespace)
4	MEDIUM	GPU passthrough (`--gpu`) bind-mounts all `/dev/nvidia*` and `/dev/dri/*` into container
5	MEDIUM	Missing IPC namespace isolation (`CLONE_NEWIPC` not set) — shared memory leakage between container and host
6	MEDIUM	TAP networking requires `CAP_NET_ADMIN` in VM runtime
7	MEDIUM	cgroup v2 memory limits require delegation (often root)
8	MEDIUM	WSL overlay runs as root inside WSL distribution
9-12	MEDIUM	Various `system()` -based command execution patterns in container/WSL modules
13-16	LOW	UID/GID map manipulation (standard rootless pattern), `/dev/kvm` access, `/proc` mount without `MS_NOEXEC` fallback
17-19	INFO	Standard Unix domain socket usage, pipe-based IPC, proper signal handling

---

14. Windows Deployment Analysis

Agent Risk Rating: LOW

Windows-Specific Implementation

The Windows build demonstrates **good cross-platform hygiene**:

Feature	Implementation
Process execution	`CreateProcess` + `CREATE_NO_WINDOW` for background commands; ConPTY for terminal
Network	Winsock2 with `WSAStartup` / `WSACleanup` ; WinINet for WSL image downloads
Screen capture	GDI (`BitBlt` , `GetDIBits` , `EnumDisplayMonitors`)
File transfer	`TransmitFile` (zero-copy Winsock)
Terminal	ConPTY (`CreatePseudoConsole`) on Windows 10 1809+
TLS	Static libcurl with bundled `curl-ca-bundle.crt`

Windows-Specific Findings

#	Severity	Finding
1-5	INFO	Standard Windows API usage patterns; proper `#ifdef _WIN32` / `SX_WINDOWS` guards; no Windows-specific backdoors or novel attack surfaces beyond the general codebase findings

Assessment: No additional Windows-specific vulnerabilities were identified beyond those already documented in the cross-platform audit sections. The Windows deployment surface is narrower due to the exclusion of 10 Linux/macOS-only binaries.

---

15. Telemetry and Tracking

Note: The telemetry-tracking audit branch was not available for merge. Assessment is based on static analysis of telemetry-related binaries identified in other audit reports.

Identified Telemetry Components

Binary	Endpoint	Data Sent
`scorpiox-usage`	`https://code.scorpiox.net/usage-send`	Token usage metrics
`scorpiox-emit-session`	`https://code.scorpiox.net/sessions-send`	Session event data

Observations

- Both telemetry endpoints use **HTTPS** (encrypted)
- Telemetry binaries are included in the Windows build
- No evidence of covert data collection channels beyond the two documented endpoints
- Session emit writes temporary files to `/tmp/` before transmission (see File I/O findings)

---

16. Consolidated Risk Matrix

Area	Finding	Severity	Status	Windows?	Recommendation
TLS	SSL cert verification disabled on token-fetch endpoints	CRITICAL	Open	Yes	Enable `CURLOPT_SSL_VERIFYPEER=1`, `CURLOPT_SSL_VERIFYHOST=2`; use `CURLOPT_CAINFO` for CA bundle
TLS	Token endpoints use plaintext HTTP	CRITICAL	Open	Yes	Migrate all `http://token.scorpiox.net/` to `https://`
TLS	Traffic proxy global TLS bypass	CRITICAL	Open	No	Remove `NODE_TLS_REJECT_UNAUTHORIZED=0` and similar env overrides
Cmd Injection	Hook data JSON in `system()` — single quotes unescaped	CRITICAL	Open	Yes	Use `fork()` + `execvp()` instead of `system()`
Cmd Injection	WhatsApp message injection via `system()`	CRITICAL	Open	No	Replace `system()` with `fork()` + `execvp()`
Cmd Injection	iMessage sender injection via `system()`	CRITICAL	Open	No	Replace `system()` with `fork()` + `execvp()`
Cmd Injection	`is_safe_shell_arg()` missing single-quote block	CRITICAL	Open	Yes	Add `'` to rejected characters
Memory	Unchecked `malloc` in HTTP server — DoS	CRITICAL	Open	Yes	Add NULL checks after all allocations in server path
Memory	Use-after-free in agent/protocol handlers	HIGH	Open	Yes	Audit and fix free/use ordering
Memory	35 unsafe `realloc` patterns	MEDIUM	Open	Yes	Use `tmp = realloc(ptr, size); if (tmp) ptr = tmp;`
Memory	92 missing NULL checks after allocation	LOW	Open	Yes	Add systematic NULL checks
Network	9 plaintext HTTP token endpoints	HIGH	Open	Yes	Enforce HTTPS-only
Network	5 hardcoded internal IP addresses	MEDIUM	Open	Yes	Remove from distributed config; use empty defaults
Build	Missing compiler hardening flags	MEDIUM	Open	Yes	Add `-fstack-protector-strong -D_FORTIFY_SOURCE=2`
Credentials	Placeholder API key `"xxx"` triggers auth attempts	MEDIUM	Open	Yes	Set default to empty string `""`
Credentials	SSH `root` user + internal IP in shipped config	MEDIUM	Open	Yes	Ship config with empty values
File I/O	Sensitive env vars logged in plaintext	HIGH	Open	Yes	Apply `sx_config_is_sensitive()` filtering
File I/O	Predictable temp files without `mkstemp`	HIGH	Open	Yes	Use `mkstemp()` / `tmpfile()` for temp file creation
File I/O	Missing `O_CLOEXEC` on file descriptors	MEDIUM	Open	Yes	Add `O_CLOEXEC` to all `open()` calls
Privilege	WSL runtime executes as root	HIGH	Open	Yes	Use non-root WSL user where possible
Privilege	Missing IPC	MEDIUM	Open	No	Add `CLONE_NEWIPC` to clone flags

	namespace isolation				
Supply Chain	Pre-built ELF binaries in repository	MEDIUM	Open	No	Build from source in CI
Buffer	Signed `int` accumulators in `snprintf()` loops	MEDIUM	Open	Yes	Use `size_t` with bounds clamping

---

17. Conclusion & Recommendations

Overall Assessment

ScorpioX Code demonstrates a **mature C codebase** with strong fundamentals — zero unsafe string functions, comprehensive static linking, no privilege escalation vectors, and careful namespace isolation in container components. The HEAD commit shows active security remediation.

However, **three systemic issues** elevate the overall risk to **HIGH**:

1. **TLS verification bypass** is the most urgent issue — authentication tokens are transmitted in cleartext and certificate verification is disabled on critical endpoints.
2. **Residual command injection** via `system()` / `popen()` calls remains widespread despite recent fixes.
3. **Memory safety gaps** (unchecked allocations, unsafe realloc patterns) are typical of C codebases but create denial-of-service risk in network-facing components.

Priority Remediation

Priority	Action	Impact
P0 — Immediate	Enable TLS certificate verification on all token-fetch endpoints	Prevents credential interception
P0 — Immediate	Migrate token URLs from `http://` to `https://`	Encrypts authentication traffic
P1 — High	Replace remaining `system()` / `popen()` with `fork()` + `execvp()`	Eliminates command injection class
P1 — High	Fix `is_safe_shell_arg()` single-quote bypass	Closes escaping gap
P2 — Medium	Add NULL checks after allocations in server/agent paths	Prevents DoS via NULL dereference
P2 — Medium	Enable compiler hardening flags	Defense-in-depth against memory corruption
P2 — Medium	Use `mkstemp()` for temp file creation	Prevents symlink/race attacks
P3 — Low	Remove internal IPs from distributed config	Reduces information leakage
P3 — Low	Fix unsafe `realloc` patterns	Prevents memory leaks under pressure

Windows Workstation Risk Rating

For the scoped Windows workstation deployment: **MEDIUM-HIGH** — The most severe findings (TLS

bypass, credential plaintext transmission, hook command injection) apply to Windows. However, the highest-CVSS command injection vectors (WhatsApp/iMessage message injection) are Linux/macOS-only.

---

18. Appendix: Audit Methodology

Audit Framework

This audit employed **11 specialized automated agents**, each analyzing the complete codebase from a specific security domain:

Agent	Domain	Method
`supply-chain`	Dependencies, vendored code, package managers	Dependency tree analysis, license check, binary detection
`build-provenance`	Build system, compiler flags, code signing	CMake/Makefile analysis, flag inventory
`network-endpoints`	URLs, IPs, ports, protocols	Static string extraction, protocol classification
`tls-security`	TLS configuration, certificate verification	libcurl option audit, mbedTLS config review
`credential-hardcode`	API keys, passwords, tokens, secrets	Pattern matching, entropy analysis, config review
`file-io`	File operations, temp files, permissions	Syscall audit, permission analysis, race condition detection
`buffer-safety`	Buffer overflows, format strings, bounds checking	Function census, pattern analysis, accumulator audit
`memory-safety`	malloc/free correctness, use-after-free, leaks	Allocation tracking, free-order analysis, realloc patterns
`command-injection`	system()/popen()/exec() with user input	Taint analysis, shell metacharacter flow
`privilege-access`	Setuid, namespaces, capabilities, process spawning	Privilege API audit, namespace isolation review
`windows-deployment`	Windows binaries, APIs, platform-specific risks	Binary inventory, Win32 API audit, cross-platform guard review

Scope

- **In Scope:** All non-vendor `.c` and `.h` source files (150 files, 170,860 lines)
- **Vendor Analysis:** Vendored libraries (Mbed TLS, yyjson) reviewed for known vulnerabilities and configuration issues
- **Out of Scope:** Runtime behavior, dynamic analysis, penetration testing, third-party API security

Severity Classification

Level	Definition
CRITICAL	Exploitable remotely without authentication; direct credential/data compromise
HIGH	Significant security weakness; requires specific conditions for exploitation
MEDIUM	Security concern with mitigating factors; defense-in-depth gap

LOW	Minor issue; best-practice deviation with minimal direct impact
INFO	Observation or positive finding; no direct security impact