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: `28b0b1b7dc2738c86102c359b61e395b54b4ee19`
Classification: CONFIDENTIAL — For Corporate Review Only

---

1. Executive Summary

This report consolidates findings from 13 specialized security audit agents that performed a comprehensive review of the ScorpioX Code codebase at commit `28b0b1b`. The software is an AI-powered development CLI platform written entirely in C (approximately 402,541 lines of code: 151,596 project code, 228,526 vendored libraries), targeting Windows, Linux, and macOS.

Overall Risk Rating: MEDIUM

The codebase demonstrates strong foundational security practices — zero external package manager dependencies at build time, fully static linking on Linux, 99.6% safe buffer function usage (2,227 `snprintf` vs 1 `sprintf`), no setuid/setgid usage, no executable memory regions, and all telemetry disabled by default. However, meaningful risks exist in TLS security (plaintext token fetch endpoints, silent STARTTLS fallback), memory safety (57+ unchecked malloc sites), distribution integrity (no code signing, graceful SHA256 degradation), and hardcoded credentials in documentation/automation files.

Findings Summary

Severity	Count	Windows-Scoped
Critical	7	7
High	20	20
Medium	89	80
Low	55	52
Info	103	97
Total	**274**	**159** (excl. info)

Key Strengths Identified:

- Zero external package manager dependencies for the core C build
- Two vendored C libraries only (yyjson 0.12.0 MIT, mbedTLS 3.6.6 LTS)
- Single-organization commit provenance across 1,096 commits
- Full static linking on Linux (musl libc)
- Stack protector, FORTIFY_SOURCE=2, and control-flow protection enabled
- 99.6% bounded string operations (`snprintf` / `strncpy` vs unsafe alternatives)
- No privilege escalation code (zero setuid/setgid/capabilities calls)

- All telemetry/tracking disabled by default (opt-in only)
- No mandatory data collection
- Rootless container runtime with proper namespace isolation
- API keys redacted in log output via `sx_config_is_sensitive()`

Key Risks Identified:

- Token fetch endpoints default to plaintext HTTP (Critical)
- TCP token fetch protocol has zero TLS (Critical)
- Hardcoded credentials in documentation and automation files (Critical)
- Distribution server credentials committed to repository (Critical)
- SMTP relay falls back to plaintext silently (High)
- FRP TLS client missing hostname verification (High)
- No code signing on any distributed binaries (High)
- Container install paths bypass hash verification (High)
- 57+ malloc/calloc sites without NULL return checks (High)
- MITM proxy tool globally disables TLS verification (High)

---

2. Deployment Scope — Windows Workstation

Primary Deployment Target: Windows Workstation

This audit's risk assessment is primarily scoped to **Windows workstation deployment**. The software produces 56 Windows-compatible executables and 1 optional shared library. Server-side components (container runtime, KVM VM runner) are Linux-only and are not part of the Windows deployment footprint.

Windows Binaries (56 executables + 1 DLL)

Core CLI & TUI: `sx.exe`, `scorpiox-bash.exe`, `scorpiox-vi.exe`, `scorpiox-config.exe`, `scorpiox-tmux.exe`, `scorpiox-multiplexer.exe`

AI Provider Proxies & Token Management: `scorpiox-gemini.exe`, `scorpiox-openai.exe`, `scorpiox-claudecode-fetchtoken.exe`, `scorpiox-claudecode-models.exe`, `scorpiox-claudecode-refreshtoken.exe`, `scorpiox-codex-fetchtoken.exe`, `scorpiox-codex-refreshtoken.exe`, `scorpiox-gemini-fetchtoken.exe`, `scorpiox-server-fetchtoken.exe`

Utilities: `scorpiox-agent.exe`, `scorpiox-askuserquestion.exe`, `scorpiox-beam.exe`, `scorpiox-busybox.exe`, `scorpiox-conv.exe`, `scorpiox-debug.exe`, `scorpiox-dns.exe`, `scorpiox-docs.exe`, `scorpiox-email.exe`, `scorpiox-emit-session.exe`, `scorpiox-executecurl.exe`, `scorpiox-fetch.exe`, `scorpiox-frp.exe`, `scorpiox-hook.exe`, `scorpiox-host.exe`, `scorpiox-kql.exe`, `scorpiox-logger.exe`, `scorpiox-mcp.exe`, `scorpiox-mirror-git.exe`, `scorpiox-otp.exe`, `scorpiox-planmode.exe`, `scorpiox-printlogs.exe`, `scorpiox-pwsh.exe`, `scorpiox-renderimage.exe`, `scorpiox-rewind.exe`, `scorpiox-runtest.exe`, `scorpiox-screenshot.exe`, `scorpiox-sdk.exe`, `scorpiox-search.exe`, `scorpiox-server.exe`, `scorpiox-server-email.exe`, `scorpiox-server-http2tcp.exe`, `scorpiox-skills.exe`, `scorpiox-systemprompt.exe`, `scorpiox-tasks.exe`, `scorpiox-transcript.exe`, `scorpiox-usage.exe`, `scorpiox-vault-git.exe`, `scorpiox-voice.exe`, `scorpiox-websearch.exe`, `scorpiox-wsl.exe`

Windows-Only: `scorpiox-wsl.exe` (WSL2 integration), `scorpiox-busybox.exe` (GNU tool manager)

Optional Shared Library: `libsx.dll` (opt-in build via `SX_BUILD_DLL`)

Linux/macOS-Only Binaries (NOT deployed on Windows)

Binary	Platform	Purpose
`scorpiox-unshare`	Linux only	Rootless container runtime (user namespaces)

`scorpiox-init`	Linux only	Container PID 1 init
`scorpiox-vm`	Linux only	KVM virtual machine runner
`scorpiox-sshpass`	Unix only	SSH PTY password wrapper
`scorpiox-traffic`	Unix only	Network traffic tool
`scorpiox-podman`	Unix only	Podman container wrapper
`scorpiox-cron`	Not Windows	Crontab wrapper
`scorpiox-whatsapp`	Not Windows	WhatsApp CLI bridge
`scorpiox-thunderbolt4`	macOS only	Thunderbolt 4 file transfer
`scorpiox-imessage`	macOS only	iMessage CLI

Windows-Filtered Findings

When scoped to Windows workstation deployment only (excluding findings that exclusively affect Linux/macOS-only binaries):

Severity	All Platforms	Windows Only
Critical	7	7
High	20	20
Medium	89	80
Low	55	52
Actionable Total	**171**	**159**

---

3. Changes Since Last Audit

Previous Audit: `output/2026-04-28_2b6f3d3` (commit `2b6f3d3`)
Previous Total Findings: 240

Severity	Previous	Current	Delta
Critical	17	7	**↓ 10**
High	28	20	**↓ 8**
Medium	68	89	**↑ 21**
Low	53	55	**↑ 2**
Info	74	103	**↑ 29**
Total	**240**	**274**	**↑ 34**

Analysis: Critical and high-severity findings decreased significantly (−10 and −8 respectively), indicating remediation of the most serious issues. The increase in medium/low/info findings reflects improved audit coverage — this audit uses 13 specialized agents (up from 13 in the previous round) with deeper per-domain analysis, particularly in memory safety (47 medium findings for unchecked allocations that were previously rolled up into fewer findings). The overall risk posture has **improved from HIGH to MEDIUM** despite the higher total count, because the count increase is driven by finer-grained enumeration of lower-severity issues rather than new vulnerabilities.

---

4. Supply Chain & Dependencies

Agent: supply-chain | **Risk:** LOW | **Findings:** 0C / 0H / 1M / 3L / 3I

Architecture

The project is a C-based application suite with minimal external dependencies. The supply chain attack surface is small:

- **Vendored C Libraries (2):**
  - **yyjson v0.12.0** — MIT license, ~19,556 LOC, fast JSON parser. No known CVEs.
  - **mbedTLS v3.6.6** — Apache-2.0/GPL-2.0+ dual license, ~208,584 LOC, TLS/crypto. Recent LTS release with custom minimal config (`sx_mbedtls_config.h`).
- **System Dependencies (build-time):** libcurl, OpenSSL, zlib, ncurses, pthreads — resolved by build environment, not bundled.
- **npm Sub-project (bridge/):** 2 direct deps, ~97 transitive. Lock file with integrity hashes present. Server-side component, not part of Windows deployment.
- **No** submodules, no FetchContent, no runtime package downloads in the core build.

Notable Findings

ID	Severity	Finding
SC-M1	Medium	Two pre-built ELF binaries committed to repo (`bridge/scorpiox-ws2tcp` , `bridge/scorpiox-ws2tcp-arm64`) — not reproducible from source
SC-L1	Low	mbedTLS 3.6.6 is large (~208K LOC) vendored codebase — manual update required
SC-L2	Low	npm dependency `@whiskeysockets/baileys` is a release candidate (`rc.9`)
SC-L3	Low	gnuwin64 download script fetches MSYS2 packages at build time

---

5. Build System & Code Provenance

Agent: build-provenance | **Risk:** LOW | **Findings:** 0C / 0H / 2M / 2L / 6I

Build Configuration

Property	Value
Build System	CMake ≥ 3.16
Language Standard	C11 (GNU extensions disabled)
Default Build	Release, fully static linked
Cross-compilation	ARM64/aarch64 support
Platforms	Linux, macOS, Windows (MinGW)

Compiler Security Flags

Flag	Status
`-Wall -Wextra`	✅ Enabled
`-fstack-protector-strong`	✅ Enabled

`-D_FORTIFY_SOURCE=2`	✅ Enabled (Release)
`-fcf-protection`	✅ Enabled (GCC)
`-Wl,-z,relro` / `-Wl,-z,now`	⚠ Missing
`-fPIE` / `-pie`	⚠ Missing

Code Provenance

- **1,096 commits**, single-organization authorship
- All vendored code committed in-tree (no submodules)
- Binary stripping enabled for release builds (~40+ targets)

Notable Findings

ID	Severity	Finding
BP-M1	Medium	RELRO and PIE linker hardening flags absent (limited impact for static builds)
BP-M2	Medium	Bridge Makefile missing stack protector and FORTIFY_SOURCE
BP-L1	Low	yyjson compiled with `-w` (all warnings suppressed)
BP-L2	Low	Python code generation at build time (`generate_*.py` scripts)

---

6. Network Endpoints

Agent: network-endpoints | **Risk:** MEDIUM | **Findings:** 3C / 5H / 12M / 18L / 57I

Overview

95 unique hardcoded network endpoints identified across 5 categories: external API URLs, internal infrastructure domains, hardcoded IP addresses, hardcoded ports, and exposed credentials.

Critical Findings

ID	Finding	Risk
NE-C1	Hardcoded plaintext passwords in documentation (`xboxone` for SSH/sudo on lab machines)	Credentials in git history
NE-C2	NAS/dist server SMB password in automation file (`Dd1122110909`)	Distribution infrastructure compromise
NE-C3	Credentials in build/skill automation files (`sshpass` commands)	Persistent exposure in repo

High Findings

ID	Finding	Risk
NE-H1	Hardcoded public IP `20.53.240.54` as default TCP_HOST compiled into binary	Infrastructure pinning in all binaries
NE-H2	HTTP (non-TLS) token fetch endpoints (`http://token.scorpiox.net/codex` , `/gemini`)	Plaintext token transmission
		Tight coupling to auth

NE-H3	JWT issuer/audience hardcoded in server	infrastructure

Recommendations

1. Remove all plaintext credentials from repository and rotate them immediately
2. Change token endpoints to HTTPS
3. Use DNS hostnames instead of hardcoded IP addresses
4. Make JWT issuer/audience configurable

---

7. TLS/SSL Security

Agent: tls-security | **Risk:** HIGH | **Findings:** 2C / 4H / 4M / 2L / 5I

This is the highest-risk audit domain. Multiple TLS weaknesses exist in transport security.

Critical Findings

ID	Finding	Impact
TLS-C1	Token fetch endpoints default to plaintext HTTP	API tokens interceptable via MITM
TLS-C2	TCP token fetch protocol has zero TLS support	API keys and OAuth credentials sent in cleartext over TCP

High Findings

ID	Finding	Impact
TLS-H1	SMTP relay falls back to plaintext silently when STARTTLS unsupported	STARTTLS stripping vulnerability; AUTH credentials exposed
TLS-H2	Direct MX delivery (port 25) has no STARTTLS support	Email content transmitted in plaintext
TLS-H3	FRP TLS client missing SNI hostname verification (`mbedtls_ssl_set_hostname()` never called)	Any valid CA-signed certificate accepted regardless of hostname
TLS-H4	MITM proxy tool globally disables TLS verification for child processes	Design-intentional but dangerous if misused

Medium Findings

ID	Finding
TLS-M1	WebSocket client (`sx_websocket.c`) relies on libcurl defaults for TLS — no explicit minimum version
TLS-M2	mbedTLS config enables TLS 1.2 only (no TLS 1.3) — acceptable but aging
TLS-M3	No certificate pinning for any API endpoints
TLS-M4	SMTP TLS implementation does not verify server certificate hostname

Recommendations

1. **Immediate:** Change all token fetch default URLs to `https://`
2. **Immediate:** Add TLS to TCP token fetch or restrict to loopback
3. **High Priority:** Make STARTTLS mandatory for relay connections
4. **High Priority:** Add `mbedtls_ssl_set_hostname()` call in FRP client

5. Implement opportunistic STARTTLS for direct MX delivery

---

8. Hardcoded Credentials

Agent: credential-hardcode | **Risk:** LOW | **Findings:** 0C / 0H / 3M / 2L / 2I

No real API keys or bearer tokens are embedded in compiled source code. Findings relate to placeholder values and public OAuth client IDs.

ID	Severity	Finding
HC-M1	Medium	Placeholder API key `"xxx"` hardcoded as default for OPENAI_API_KEY — may be sent in requests
HC-M2	Medium	Hardcoded Anthropic OAuth client ID (public, but not configurable)
HC-M3	Medium	Hardcoded OpenAI/Codex OAuth client ID (public, but not configurable)
HC-L1	Low	Hardcoded Azure IP `20.53.240.54` for TCP token fetch default
HC-L2	Low	Empty credential field placeholders in embedded config

Recommendations

1. Set OPENAI_API_KEY default to empty string `""`
2. Make OAuth client IDs configurable with current values as defaults

---

9. File I/O & Data Handling

Agent: file-io | **Risk:** MEDIUM | **Findings:** 0C / 1H / 5M / 5L / 0I

Positive Practices

- `mkstemp()` used for most temp files
- Sensitive config values redacted in logs via `sx_config_is_sensitive()`
- Directory creation uses `0700` permissions via `sx_mkdir()`
- Path traversal protection in HTTP server using `_fullpath()` canonicalization

Notable Findings

ID	Severity	Finding
FIO-H1	High	Predictable temp file names in HTTP server (POST body): PID + timestamp, symlink race
FIO-M1	Medium	Predictable temp file in editor (`sx-edit-<PID>.txt`)
FIO-M2	Medium	Predictable temp file in container image listing
FIO-M3	Medium	DNS stats/PID files in `/tmp` without restrictive permissions
		Config files with API keys written without

FIO-M4	Medium	explicit `0600` permissions
FIO-M5	Medium	Log files may contain sensitive headers before redaction

Recommendations

1. Replace all PID/timestamp-based temp files with `mkstemp()` pattern
2. Explicitly set `0600` permissions on config files containing API keys
3. Use `O_CREAT|O_EXCL` for PID files

---

10. Buffer Safety

Agent: buffer-safety | **Risk:** LOW | **Findings:** 0C / 2H / 2M / 3L / 2I

Safe vs Unsafe Function Ratio

Category	Count
`snprintf`	2,227
`strncpy`	594
`fgets`	108
`vsnprintf`	36
`strncat`	20
`strcat` (unsafe)	11
`sprintf` (unsafe)	1 (vendor only)
`strcpy` / `gets` / `vsprintf`	0
Safe:Unsafe ratio	**99.6% safe**

Notable Findings

ID	Severity	Finding
BUF-H1	High	`strcat` / `strncat` size_t underflow in Windows command builder loops (`scorpiox-search.c` , `scorpiox-renderimage.c`) — stack overflow when buffer nearly full
BUF-H2	High	Windows `scorpiox-bash.c` command builder uses `strcat` in loop without bounds checking
BUF-M1	Medium	Missing overflow clamping in 2 `pos += snprintf()` patterns
BUF-M2	Medium	Vendor `sprintf` fallback in mbedTLS (vendored code)

Recommendations

1. Replace `strcat` -based Windows command builders with `snprintf` -based pattern with post-call clamping
2. Add clamping to remaining `pos += snprintf()` patterns

---

11. Memory Safety

Agent: memory-safety | **Risk:** MEDIUM | **Findings:** 1C / 2H / 47M / 3L / 2I

The most finding-dense audit domain, primarily driven by a systemic pattern of unchecked malloc/calloc return values.

Critical Finding

ID	Finding
MEM-C1	Potential double-free in `sx_term.c` line 781 — `free(g_term.cap_buf)` without NULL guard in error path

High Findings

ID	Finding
MEM-H1	Unsafe `realloc` pattern loses original pointer on failure (memory leak + NULL deref)
MEM-H2	Memory leak in `sx_term.c` — allocated buffer not freed in early-return path

Medium Findings (47)

57+ allocation sites across the codebase where `malloc` / `calloc` / `realloc` return values are used without NULL checks. Key areas:

Category	Affected Files	Count
Terminal/UI	`sx_term.c` , `sxmux_vt.c`	~10 sites
Network/Provider	`sx_provider*.c` , `sx_http*.c`	~15 sites
Application/CLI	`sx.c` , `scorpiox-*.c`	~20 sites
Utilities	`sx_agent*.c` , `sx_bridge.c`	~12 sites

Recommendations

1. Add a project-wide `sx_malloc()` / `sx_calloc()` wrapper that checks for NULL and logs/aborts on failure
2. Fix the double-free in `sx_term.c` with NULL guard
3. Fix the unsafe `realloc` pattern to preserve the original pointer
4. Address the memory leak in the early-return path

---

12. Command Injection

Agent: command-injection | **Risk:** MEDIUM | **Findings:** 0C / 1H / 3M / 3L / 2I

Overview

~50+ `system()` calls, ~30+ `popen()` calls, ~40+ `exec*()` calls across 20+ source files. Many Unix paths have been migrated from `system()` to safe `fork()+execvp()` . Critical network-reachable paths are well-protected.

High Finding

ID	Finding	Platform

CJ-H1	Windows `cmd /C` CGI environment variable injection — `SX_SANITIZE_CMD` does not filter `%`, `!`, backticks, or `;`	Windows (network-reachable)

Medium Findings

ID	Finding	Platform
CJ-M1	Shell injection via single-quote in `osascript` command	macOS only
CJ-M2	`osascript` voice command injection via unsanitized user text	macOS only
CJ-M3	`system()` call in OTP generator with partially-sanitized input	Cross-platform

Recommendations

1. **Immediate:** Add `%`, `!`, `` ` ``, `;`, `(`, `)`, newlines to `SX_SANITIZE_CMD` filter; better yet, use `CreateProcess()` on Windows instead of `cmd /C`
2. Use `execvp()` with argument arrays instead of `system()` for remaining shell-based calls
3. Properly escape single quotes in osascript commands (macOS)

---

13. Privilege & Access Control

Agent: privilege-access | **Risk:** MEDIUM | **Findings:** 0C / 1H / 4M / 3L / 7I

Positive Security Posture

- **Zero** setuid/setgid/capability calls
- **Zero** SUID/SGID bits in build system
- No privilege escalation code
- Rootless container runtime (uses `CLONE_NEWUSER` for non-root)
- 30+ process spawning sites use safe `fork()+execvp()` pattern

High Finding

ID	Finding
PA-H1	Container runtime uses unprivileged `ioctl(LOOP_CTL_GET_FREE)` — may fail without loopback device access, causing fallback to less-isolated paths

Medium Findings

ID	Finding
PA-M1	Root detection in container runtime skips user namespace isolation when running as root
PA-M2	HTTP server binds to `0.0.0.0` by default — exposes script execution to network
PA-M3	DNS server on port 53 requires elevated privileges on most systems
PA-M4	Several `system()` calls remain in non-critical utility paths

Recommendations

1. Default HTTP server to `127.0.0.1` binding

2. Document privilege requirements for DNS server
3. Continue migration from `system()` to `execvp()` for remaining call sites

---

14. Windows Deployment Analysis

Agent: windows-deployment | **Risk:** LOW | **Findings:** 0C / 0H / 2M / 6L / 8I

Windows Build Characteristics

- Cross-compiled via MinGW with static linking
- Uses standard Win32 APIs (`CreateProcess` , `WriteFile` , ConPTY, WinINet, named pipes)
- No Windows registry manipulation
- No Windows service installation
- No privilege escalation mechanisms
- Path traversal protection via `_fullpath()` canonicalization

Medium Findings

ID	Finding
WD-M1	`scorpiox-server` listens on all interfaces by default (0.0.0.0:8080) — Windows Firewall should prompt but permissive configs expose Python script execution to LAN
WD-M2	`scorpiox-dns` binds to port 53 — requires admin privileges on Windows, may conflict with DNS Client service

Low Findings

ID	Finding
WD-L1	Config stored in `%USERPROFILE%\.scorpiox\` with API keys in plaintext `scorpiox-env.txt`
WD-L2	`scorpiox-bash` creates temporary `.bat` files (cleaned up after execution)
WD-L3	`scorpiox-server-fetchtoken` listens on port 9800 for TCP connections
WD-L4	`scorpiox-server-email` runs SMTP/IMAP server (blocked by firewall by default)
WD-L5	`scorpiox-beam` uses raw TCP for file transfer without encryption
WD-L6	`scorpiox-bash` uses `CreateProcess` with named pipes — injection mitigated by temp `.bat` approach

---

15. Telemetry and Tracking

Agent: telemetry-tracking | **Risk:** LOW | **Findings:** 0C / 0H / 0M / 2L / 5I

Verdict: No Tracking by Default

Both telemetry features (`USAGE_TRACKING` and `EMIT_SESSION_TRACKING`) are **disabled by default** across all configuration sources — compiled-in defaults, embedded WASM config, and the shipped environment file. No network telemetry calls are made unless the user explicitly opts in.

Data Collection Inventory

When `USAGE_TRACKING=1` (opt-in): Collects session ID, provider, model, token counts, rate limit info,

hostname, username, OS, architecture, project name, git branch, version. **Does NOT collect:** IP address, MAC address, device UUID, file contents, conversation text.

When `EMIT_SESSION_TRACKING=1` (opt-in): Collects full conversation text (user prompts, AI responses, tool output), session metadata, timestamps. This is the most privacy-sensitive feature but is disabled by default and clearly labeled.

Low Findings

ID	Finding
TT-L1	`USAGE_TRACKING` collects hostname and username — PII when enabled
TT-L2	`EMIT_SESSION_TRACKING` transmits full conversation content — significant privacy impact when enabled

Recommendations

1. Add a privacy notice when users enable `EMIT_SESSION_TRACKING`
2. Consider hashing hostname/username before transmission in usage tracking
3. Document data retention policies for collected telemetry

---

16. Install Script & Distribution Security

Agent: install-script | **Risk:** HIGH | **Findings:** 1C / 4H / 4M / 3L / 4I

Critical Finding

ID	Finding
IS-C1	NAS/distribution server credentials (SMB username + password) hardcoded in repository — anyone with repo access can push arbitrary binaries

High Findings

ID	Finding
IS-H1	SHA256 verification skipped when `.sha256` sidecar unavailable (graceful degradation weakens integrity)
IS-H2	Container install path has zero integrity verification (downloads and extracts without hash checks)
IS-H3	No code signing on any distributed binaries (no GPG, cosign, Authenticode, or Apple notarization)
IS-H4	WhatsApp bridge release lacks SHA256 sidecar generation

Medium Findings

ID	Finding
IS-M1	`curl -k` (TLS verification disabled) used in documented install commands
IS-M2	macOS codesign uses ad-hoc signature (`--sign -`), not a verified identity
IS-M3	Self-update uses file-size comparison as primary change detection
IS-M4	Non-atomic install in container paths (partial state on interruption)

Recommendations

1. **Immediate:** Rotate NAS/distribution server credentials and remove from repository
2. **High Priority:** Implement Authenticode signing for Windows binaries
3. **High Priority:** Make SHA256 verification mandatory (fail-closed instead of graceful degradation)
4. Add integrity verification to container install paths
5. Remove `curl -k` from all documentation

---

17. Consolidated Risk Matrix

#	Area	Finding	Severity	Status	Recommendation
1	TLS	Token fetch endpoints default to HTTP	Critical	Open	Change default URLs to HTTPS
2	TLS	TCP token fetch has zero TLS	Critical	Open	Add TLS or restrict to loopback
3	Network	Plaintext passwords in documentation	Critical	Open	Remove from repo, rotate credentials
4	Network	NAS/dist server SMB password in repo	Critical	Open	Rotate credentials, remove from repo
5	Network	Credentials in automation files	Critical	Open	Use SSH keys or vault integration
6	Install	Distribution server credentials in repo	Critical	Open	Rotate and remove (same as #4)
7	Memory	Double-free in terminal error path	Critical	Open	Add NULL guard before free
8	TLS	SMTP relay falls back to plaintext silently	High	Open	Make STARTTLS mandatory
9	TLS	MX delivery has no STARTTLS	High	Open	Implement opportunistic STARTTLS
10	TLS	FRP client missing hostname verification	High	Open	Add `mbedtls_ssl_set_hostname()`
11	TLS	MITM proxy disables TLS verification	High	Open	Document risk, add warnings
12	File I/O	Predictable temp files in HTTP server	High	Open	Use `mkstemp()` pattern
13	Buffer	Windows command builder stack overflow	High	Open	Replace with `snprintf`-based builder
14	Buffer	Windows bash command builder overflow	High	Open	Replace with `snprintf`-based builder
15	Memory	Unsafe realloc loses original pointer	High	Open	Preserve pointer before realloc
16	Memory	Memory leak in terminal early-return	High	Open	Free buffer in error path
17	Install	SHA256 verification graceful	High	Open	Make verification mandatory

		degradation			
18	Install	Container install bypasses hash verification	High	Open	Add integrity checks
19	Install	No code signing on any binaries	High	Open	Implement Authenticode/GPG signing
20	Install	WhatsApp bridge release lacks SHA256	High	Open	Generate sidecar hashes
21	Cmd Inj	Windows CGI `SX_SANITIZE_CMD` incomplete	High	Open	Add `%!;()` to filter or use CreateProcess
22	Privilege	Container ioctl fallback path	High	Open	Handle gracefully
23	Network	Hardcoded public IP in compiled default	High	Open	Use DNS hostname
24	Network	JWT issuer/audience hardcoded	High	Open	Make configurable
25	Memory	57+ unchecked malloc/calloc sites	Medium	Open	Add project-wide allocation wrapper
26	Build	RELRO/PIE linker flags missing	Medium	Open	Add to linker flags
27	Build	Bridge Makefile missing hardening flags	Medium	Open	Add stack protector, FORTIFY_SOURCE
28	Credential	Placeholder `"xxx"` for OPENAI_API_KEY	Medium	Open	Set default to empty string
29	File I/O	Predictable editor temp file	Medium	Open	Use `mkstemp()`
30	File I/O	Config files written without `0600`	Medium	Open	Set explicit permissions
31	Cmd Inj	OTP generator with `system()`	Medium	Open	Use `execvp()`
32	Privilege	HTTP server binds 0.0.0.0 by default	Medium	Open	Default to 127.0.0.1
33	Windows	DNS server port 53 requires admin	Medium	Open	Document requirements
34	Install	`curl -k` in documented commands	Medium	Open	Remove insecure flag
35	Install	Ad-hoc macOS codesign	Medium	Open	Use verified identity
36	Telemetry	Hostname/username collected in usage	Low	Open	Hash before transmission
37	Telemetry	Full conversation in session tracking	Low	Open	Add privacy notice
38	Supply Chain	Pre-built ELF binaries in repo	Medium	Open	Build from source in CI

18. Conclusion & Recommendations

Overall Assessment

ScorpioX Code demonstrates **strong foundational security engineering** for a C codebase of this scale. The zero-dependency architecture, comprehensive use of bounded string functions (99.6%), absence of privilege escalation code, and opt-in-only telemetry represent mature security practices. The static linking strategy on Linux eliminates an entire class of shared library attacks.

Priority Remediation Roadmap

Immediate (Critical — address before deployment): 1. Rotate all hardcoded credentials and remove from repository history 2. Change token fetch endpoints to HTTPS 3. Add TLS to TCP token fetch protocol or restrict to loopback 4. Fix double-free in `sx_term.c`

Short-Term (High — address within 30 days): 5. Make STARTTLS mandatory for SMTP relay 6. Add hostname verification to FRP TLS client 7. Replace all predictable temp file names with `mkstemp()` 8. Fix Windows command builder buffer overflows 9. Implement code signing for distributed binaries (Authenticode for Windows) 10. Make SHA256 verification mandatory (fail-closed)

Medium-Term (Medium — address within 90 days): 11. Add project-wide `sx_malloc()` wrapper with NULL checking 12. Add RELRO/PIE linker flags 13. Continue `system()` to `execvp()` migration 14. Enhance `SX_SANITIZE_CMD` filter for Windows 15. Default HTTP server to localhost binding

Long-Term (Low — address as resources allow): 16. Implement certificate pinning for critical API endpoints 17. Add TLS 1.3 support to mbedTLS configuration 18. Evaluate reproducible build infrastructure 19. Hash PII fields in telemetry before transmission

Windows Workstation Deployment Recommendation

APPROVED WITH CONDITIONS — The software is suitable for Windows workstation deployment provided: 1. All Critical findings are addressed (credential rotation, HTTPS endpoints) 2. Windows Firewall rules are configured to block unexpected listening ports 3. Users are informed that `scorpiox-server` and `scorpiox-dns` bind to network interfaces 4. Configuration directory permissions are verified on NTFS

---

19. Appendix: Audit Methodology

Audit Framework

This security review was conducted using 13 specialized automated audit agents, each focused on a specific security domain:

#	Agent	Scope
01	supply-chain	Package managers, vendored libs, pre-built binaries
02	build-provenance	Compiler flags, build system, code provenance
03	network-endpoints	Hardcoded URLs, IPs, ports, credentials in source
04	tls-security	Certificate verification, protocol versions, cipher suites
05	credential-hardcode	API keys, passwords, tokens in compiled

		code
06	file-io	Temp files, permissions, path traversal, data at rest
07	buffer-safety	Buffer overflows, format strings, integer overflows
08	memory-safety	NULL checks, use-after-free, double-free, leaks
09	command-injection	system(), popen(), exec*() with unsanitized input
10	privilege-access	setuid, capabilities, namespace isolation, process spawning
11	windows-deployment	Windows-specific binaries, APIs, risks
12	telemetry-tracking	Data collection, opt-in/opt-out, privacy
13	install-script	Distribution, integrity verification, code signing

Scope

- **In Scope:** All C source files (excluding vendored code for most agents), build system, documentation, automation scripts, distribution mechanisms
- **Out of Scope:** Runtime behavior testing, penetration testing, third-party service security, vendored library internals (except for known CVE checks)
- **Codebase:** 402,541 total LOC (151,596 project, 228,526 vendor, remainder build/docs)
- **Files Analyzed:** 149–366 C source files per agent (vendor-excluded)

Severity Definitions

Severity	Definition
Critical	Exploitable vulnerability with high impact; immediate remediation required
High	Significant security weakness; remediation within 30 days
Medium	Security concern requiring attention; remediation within 90 days
Low	Minor issue or hardening opportunity; address as resources allow
Info	Informational observation; no action required