Third-Party Software Security Review — SCORPIOX CODE

Field	Value
Software	SCORPIOX CODE
Type	AI-Powered Development Tool / CLI Platform
Language	Pure C (zero external dependencies)
Audit Date	2026-04-30
Codebase Commit	`fe4a27dca4c7a52f1735792badc534530cd24a84`
Classification	CONFIDENTIAL — For Corporate Review Only

---

1. Executive Summary

This report consolidates findings from 14 specialized security audit agents examining the SCORPIOX CODE codebase at commit `fe4a27d`. The audit covers supply chain integrity, build provenance, network security, TLS configuration, credential handling, file I/O safety, buffer safety, memory safety, command injection, privilege management, Windows deployment scope, telemetry practices, install script security, and the WPF launcher application.

Key Metrics

Metric	Value
Lines of Code	376,275 total (148,135 project / 228,140 vendor)
Total Findings	155
Critical	0
High	0
Medium	14
Low	44
Informational	97
Overall Risk Rating	**LOW**
Windows-Filtered Findings	138 (0 Critical, 0 High, 10 Medium, 38 Low, 90 Info)

Assessment Summary

SCORPIOX CODE demonstrates **strong security engineering discipline** across a 148K-line pure-C codebase. The project achieves a safe-to-unsafe string function ratio of 3,464:1, enforces centralized TLS verification by default, uses `fork()+execvp()` over `system()` for process spawning, vendors only two well-maintained C libraries (mbedTLS 3.6.6, yyjson 0.12.0), and ships zero third-party runtime dependencies in its Windows distribution. Both telemetry systems are disabled by default with no mandatory data collection.

No critical or high-severity findings were identified. The 14 medium-severity findings are concentrated in three areas: (1) file I/O edge cases on multi-user Linux systems, (2) command injection in the Linux-only container init process (requires filesystem control), and (3) install script integrity gaps (same-origin checksums without independent code signing). Most medium findings do not apply to the primary Windows workstation deployment.

---

2. Deployment Scope — Windows Workstation vs Server-Side

2.1 Primary Deployment: Windows Workstation

The following binaries are compiled for and deployed on Windows:

Binary	Purpose
`sx.exe`	Main TUI — AI coding assistant
`scorpiox.exe`	Symlink to sx.exe
`scorpiox-bash.exe`	Shell command execution (PowerShell/cmd.exe)
`scorpiox-busybox.exe`	Embedded Unix utilities
`scorpiox-screenshot.exe`	Screenshot capture
`scorpiox-vi.exe`	Minimal vi text editor
`scorpiox-config.exe`	Configuration manager
`scorpiox-websearch.exe`	Multi-engine web search
`scorpiox-fetch.exe`	URL content fetcher
`scorpiox-renderimage.exe`	Image rendering/display
`scorpiox-pwsh.exe`	PowerShell integration
`scorpiox-wsl.exe`	WSL integration
`scorpiox-tmux.exe`	Terminal session manager
`scorpiox-multiplexer.exe`	Terminal multiplexer
`scorpiox-voice.exe`	Voice recording + Whisper
`scorpiox-server.exe`	HTTP server for script execution
`scorpiox-otp.exe`	TOTP/HOTP generator
`scorpiox-transcript.exe`	Conversation viewer
`scorpiox-conv.exe`	Conversation manager
`scorpiox-rewind.exe`	Conversation checkpoint rewind
`scorpiox-debug.exe`	Diagnostics viewer
`scorpiox-logger.exe`	Log file viewer
`scorpiox-printlogs.exe`	Log export utility
`scorpiox-systemprompt.exe`	System prompt generator
`scorpiox-search.exe`	Code search (grep/rg/ag)
`scorpiox-agent.exe`	Sub-agent spawner
`scorpiox-tasks.exe`	Task manager
`scorpiox-skills.exe`	Skill manager
`scorpiox-planmode.exe`	Plan mode manager

`scorpiox-askuserquestion.exe`	Interactive question TUI
`scorpiox-gemini.exe`	Gemini provider
`scorpiox-mcp.exe`	Model Context Protocol
`scorpiox-beam.exe`	Beam sharing
`scorpiox-host.exe`	Host information
`scorpiox-openai.exe`	OpenAI provider
`scorpiox-email.exe`	Email client
`scorpiox-usage.exe`	Usage reporting (opt-in)
`scorpiox-emit-session.exe`	Session telemetry (opt-in)
`scorpiox-frp.exe`	Fast reverse proxy
`scorpiox-hook.exe`	Git hook manager
`scorpiox-vault-git.exe`	Git vault
`scorpiox-mirror-git.exe`	Git mirror
`scorpiox-dns.exe`	DNS server
`scorpiox-kql.exe`	KQL query engine
`scorpiox-sdk.exe`	SDK utilities
`scorpiox-server-email.exe`	Email server
`scorpiox-server-http2tcp.exe`	HTTP-to-TCP bridge
`scorpiox-claudecode-fetchtoken.exe`	Claude token fetcher
`scorpiox-claudecode-refreshtoken.exe`	Claude token refresh
`scorpiox-codex-refreshtoken.exe`	Codex token refresh
`scorpiox-claudecode-models.exe`	Claude model list
`scorpiox-codex-fetchtoken.exe`	Codex token fetcher
`scorpiox-gemini-fetchtoken.exe`	Gemini token fetcher
`scorpiox-server-fetchtoken.exe`	Server token fetcher
`libsx.dll`	Shared library for WPF P/Invoke

Total: 55 Windows binaries (53 executables + 1 DLL + 1 symlink)

2.2 Linux/Unix-Only Binaries (NOT deployed on Windows)

Binary	Reason
`scorpiox-sshpass`	Unix PTY/signal handling
`scorpiox-traffic`	Unix raw sockets
`scorpiox-docs`	fork/pipe-based browser
`scorpiox-runtest`	fork()+execvp() test runner
`scorpiox-executecurl`	fork-based curl executor
`scorpiox-podman`	Podman container manager
`scorpiox-unshare`	Rootless container runtime (user namespaces)
`scorpiox-init`	Container init (PID 1)
`scorpiox-vm`	KVM virtual machine runner

`scorpiox-cron`	Crontab wrapper
`scorpiox-whatsapp`	WhatsApp CLI (fork/pipe/select)

2.3 Windows-Filtered Findings Summary

Of the 155 total findings, **17 are Linux/Unix-only** and do not apply to Windows workstation deployments:

Excluded Finding	Report	Severity	Reason
NET-03: Container registry TLS verify	Network	LOW	Podman/container Linux-only
FIO-03: /tmp/sxmux predictable path	File I/O	MEDIUM	Unix socket path
MEM-02: scorpiox-unshare stack leak	Memory	MEDIUM	Linux-only binary
CJ-01: scorpiox-init descriptor injection	Command Inj.	MEDIUM	Linux-only binary
CJ-02: scorpiox-init INIT_TOOLS injection	Command Inj.	MEDIUM	Linux-only binary
PA-01: Missing CLONE_NEWIPC	Privilege	LOW	Linux namespaces
PA-02: Seccomp minimal blocklist	Privilege	LOW	Linux seccomp
PA-03: –privileged flag	Privilege	INFO	Linux container
PA-04: PR_SET_NO_NEW_PRIVS	Privilege	INFO	Linux-only
PA-05: Environment clearing in containers	Privilege	INFO	Linux-only
PA-07: Capability dropping	Privilege	INFO	Linux-only
PA-08: pivot_root	Privilege	INFO	Linux-only
PA-09: Overlayfs	Privilege	INFO	Linux-only
IS-04: –no-verify flag	Install	LOW	Linux/macOS script
IS-07: macOS ad-hoc codesign	Install	LOW	macOS only
IS-09: curl|bash pattern	Install	LOW	Linux/macOS
Finding on bridge/ws2tcp.c traversal	File I/O	(partial)	Linux-only component

Windows deployment finding count: 138 (0 Critical, 0 High, 10 Medium, 38 Low, 90 Info)

---

3. Changes Since Last Audit

Previous Audit: `output/2026-04-30_6851bc0` (commit `6851bc0`, 2026-04-30)

Metric	Previous (`6851bc0`)	Current (`fe4a27d`)	Delta
Total Findings	225	155	**−70 (↓31%)**
Critical	2	0	**−2**
High	6	0	**−6**
Medium	38	14	**−24**
Low	55	44	**−11**
Info	124	97	**−27**
Windows Findings	67	138*	**+71**

Windows Findings	67	138*	+71
Overall Risk	LOW-MEDIUM	LOW	**Improved**

**Note: The Windows finding count increase reflects improved granularity in the Windows deployment analysis (14 agents vs 13 previously, with the addition of the WPF launcher audit contributing 22 findings). The actual security posture improved — all Critical and High findings were remediated.*

Key Improvements Since Last Audit

1. **All 2 Critical findings eliminated** — zero critical findings remain
2. **All 6 High-severity findings eliminated** — zero high findings remain
3. **Medium findings reduced by 63%** (38 → 14)
4. **Commit `fe4a27d` specifically addresses** Windows command injection findings CJ-16, CJ-17, CJ-18 (per commit message: "security: fix 3 Windows command injection findings")
5. **Overall risk rating improved** from LOW-MEDIUM to LOW

---

4. Supply Chain & Dependencies

Agent: supply-chain | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 0 Medium, 1 Low, 5 Info

Key Findings

The project has an exceptionally minimal supply chain for a software platform of this scale:

- **Vendored libraries:** Only 2 — mbedTLS 3.6.6 (LTS, Apache-2.0/GPL-2.0+) and yyjson 0.12.0 (MIT)
- **No package manager dependencies** in the core C build (no npm, pip, cargo, go modules)
- **No pre-built binaries** in the source repository
- **No FetchContent, ExternalProject, or git submodules**
- **Windows release:** Exclusively native binaries (.exe, .dll) — no interpreted runtimes

An optional WhatsApp bridge component (`bridge/`) uses Bun/TypeScript with 2 npm dependencies, but this is **not** part of the core build system or Windows distribution.

Finding	Severity	Description
SC-01	LOW	Vendored library versions not tracked in a dedicated manifest file (versions are in source headers)
SC-02–SC-06	INFO	Positive observations: current library versions, proper licensing, minimal config, no pre-built binaries, zero-dependency Windows distribution

---

5. Build System & Code Provenance

Agent: build-provenance | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 0 Medium, 2 Low, 5 Info

Build Security Highlights

- **Compiler hardening:** `-Wall -Wextra -fstack-protector-strong -D_FORTIFY_SOURCE=2 -fcf-protection` (GCC)
- **Linker hardening:** Full RELRO (`-Wl,-z,relro,-z,now`), non-executable stack (`-Wl,-z,noexecstack`)
- **Static linking** by default (`SX_STATIC_LINK=ON`)

- **1,225 commits** with traceable history
- **Model header generation frozen** — Python code generator not run in CI, header checked into source

Finding	Severity	Description
BP-01	LOW	`bridge/Makefile` missing linker hardening flags (RELRO, noexecstack)
BP-02	LOW	PIE not explicitly enabled for static builds (typical for static binaries)
BP-03–BP-07	INFO	Positive: hardening flags present, deterministic build, frozen code generation, comprehensive CMake structure

---

6. Network Endpoints

Agent: network-endpoints | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 0 Medium, 3 Low, 7 Info

Network Architecture

78 unique hardcoded endpoints across 43 source files. All external connections use HTTPS. Endpoints fall into three categories:

1. **First-party infrastructure** (`*.scorpiox.net`) — distribution, token services, API proxying
2. **Third-party LLM APIs** — Anthropic, OpenAI, Google AI, Z.AI (all HTTPS)
3. **Localhost services** — HTTP for local IPC only

Finding	Severity	Description
NET-01	LOW	Hardcoded public IP (`20.53.240.54`) for TCP token transport in WASM config; native config uses hostname
NET-02	LOW	DNS server binds `0.0.0.0:53` by default (intentional; configurable via `DNS_LISTEN`)
NET-03	LOW	Container registry TLS verification defaults to `false` (Linux-only, Podman context)
NET-04–NET-10	INFO	Positive observations: all HTTPS, proper localhost binding, standard API endpoints

---

7. TLS/SSL Security

Agent: tls-security | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 0 Medium, 3 Low, 9 Info

TLS Architecture

All libcurl HTTPS connections route through a centralized `sx_curl_set_tls()` helper enforcing certificate verification by default. All mbedTLS contexts enforce TLS 1.2 minimum. The SMTP server correctly requires STARTTLS before AUTH on submission port 587.

Finding	Severity	Description
TLS-01	LOW	`SX_TLS_VERIFY` config allows disabling certificate verification (default: enabled, with warning)

TLS-02	LOW	Opportunistic STARTTLS for MX delivery uses `VERIFY_OPTIONAL` (standard SMTP behavior per RFC 3207)
TLS-03	LOW	TLS session tickets not explicitly disabled (default mbedTLS behavior, minimal risk)
TLS-04–TLS-12	INFO	Positive: centralized TLS config, fail-closed on CA errors, STARTTLS enforcement on 587, consistent `sx_curl_set_tls()` usage across all callsites, plaintext HTTP only for localhost

---

8. Hardcoded Credentials

Agent: credential-hardcode | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 2 Medium, 3 Low, 3 Info

Assessment

No actual secret values (API keys, passwords, tokens) were found hardcoded in the source code. All sensitive configuration fields use empty-string defaults and are populated at runtime via environment variables or configuration file.

Finding	Severity	Description
CRED-01	MEDIUM	Hardcoded OAuth client IDs for OpenAI/Anthropic (public IDs, no client_secret; rotation requires rebuild)
CRED-02	MEDIUM	Hardcoded infrastructure IP `20.53.240.54` in WASM config (infrastructure exposure, not a credential)
CRED-03	LOW	Hardcoded internal service URLs in embedded config
CRED-04	LOW	Embedded config key placeholders (empty-string defaults)
CRED-05	LOW	Base64-encoded SMTP challenge strings (protocol constants, not secrets)
CRED-06–08	INFO	Config architecture observations — centralized, runtime-populated

---

9. File I/O & Data Handling

Agent: file-io | **Risk:** MEDIUM | **Findings:** 0 Critical, 0 High, 3 Medium, 5 Low, 5 Info

File I/O Security Controls

- Centralized `sx_fopen()` sets `FD_CLOEXEC` on all file descriptors
- Consistent `mkstemp()` / `mkdtemp()` usage (31 call sites)
- `sx_config_is_sensitive()` redacts API keys/tokens in logs
- Config files written with `chmod(0600)`, directories with `mkdir(0700)`
- Traffic log headers redacted (x-api-key, Authorization)

Finding	Severity	Description
		API request/response bodies logged in plaintext to `api.log` (file is 0600;

FIO-01	MEDIUM	sensitive conversation content persists on disk)
FIO-02	MEDIUM	Path traversal check uses `strstr(.., "..")` — double-encoding bypass possible in `ws2tcp.c` (Linux-only) and `scorpiox-server.c`
FIO-03	MEDIUM	Predictable socket path `/tmp/sxmux` without UID qualification on fallback (**Linux-only**)
FIO-04–08	LOW	Log rotation absent, no automatic log expiry, temp file cleanup on error paths
FIO-09–13	INFO	Positive: FD_CLOEXEC, mkstemp, sensitive redaction, 0600 permissions, path traversal checks present

---

10. Buffer Safety

Agent: buffer-safety | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 0 Medium, 3 Low, 6 Info

Buffer Safety Metrics

Function	Count	Safety
`snprintf`	2,228	✅ Safe
`strncpy`	601	✅ Safe
`memcpy` (bounded)	514	✅ Safe
`fgets`	107	✅ Safe
`sprintf`	1	⚠ Unsafe
`strcpy` / `strcat` / `gets`	0	✅ Not used

Safe:Unsafe ratio = 3,464:1

Finding	Severity	Description
BUF-01	LOW	Single `sprintf` in SHA256 hex encoding (bounded by construction — 32 iterations × 2 bytes into 65-byte buffer)
BUF-02	LOW	Yamux stream buffer `uint32_t` doubling without overflow guard (requires authenticated FRP tunnel; window limits make impractical)
BUF-03	LOW	IMAP FETCH response stack buffer truncation on large envelopes (8KB buffer; silently truncates, no overflow)
BUF-04–09	INFO	Positive: zero strcpy/strcat/gets, consistent snprintf, bounded memcpy, proper fgets usage

---

11. Memory Safety

Agent: memory-safety | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 2 Medium, 2 Low, 4 Info

Memory Management Practices

The vast majority of `malloc` / `calloc` / `realloc` calls have proper NULL checks. `realloc` calls use a temporary variable pattern to avoid pointer loss.

Finding	Severity	Description
MEM-01	MEDIUM	`sx_api_init()` leaks `h->api` allocation if `h->display` calloc fails (redundant early return; one-time init path)
MEM-02	MEDIUM	`scorpiox-unshare.c` leaks `stack` (8MB) on pipe failure (**Linux-only**; process exits shortly after)
MEM-03	LOW	Terminal resize unchecked `calloc` return (graceful fallback — old buffers retained)
MEM-04	LOW	HTTP header realloc pattern correct but missing explicit overflow check on count multiplication
MEM-05–08	INFO	Positive: consistent NULL checks, proper realloc pattern, no use-after-free in critical paths, proper free-before-reassign

---

12. Command Injection

Agent: command-injection | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 2 Medium, 3 Low, 5 Info

Remediation Progress

The commit under review (`fe4a27d`) specifically fixes 3 Windows command injection findings (CJ-16, CJ-17, CJ-18). Most `system()` calls have been replaced with `fork()+execvp()` or `CreateProcess` on Windows.

Finding	Severity	Description
CJ-01	MEDIUM	`scorpiox-init` tool descriptor fields injected into shell commands (**Linux-only**; requires filesystem control of mounted volumes)
CJ-02	MEDIUM	`scorpiox-init` `INIT_TOOLS` env var passed to `sx_system_safe()` unsanitized (**Linux-only**; requires env control)
CJ-03	LOW	Windows: `SCORPIOX_INDEX_URL` config interpolated into `sx_popen_no_window()` (requires admin-level config modification)
CJ-04	LOW	`scorpiox-server.c` Python path interpolation in command string (server-side, configurable path)
CJ-05	LOW	Residual `sx_system_safe()` calls in non-critical paths (3 remaining call sites)
CJ-06–10	INFO	Positive: CreateProcess on Windows, fork+execvp on Linux, metacharacter validation, quoted arguments

---

13. Privilege & Access Control

Agent: privilege-access | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 0 Medium, 3 Low, 12 Info

Security Architecture

The container runtime (`scorpiox-unshare`) uses Linux user namespaces, pivot_root, overlayfs, and seccomp BPF. The runtime avoids setuid, uses `PR_SET_NO_NEW_PRIVS` , drops capabilities, clears environment before exec, and uses `fork+execvp` instead of `system()` .

Important context: The container runtime runs **trusted first-party agents on a private network** — not untrusted workloads. Findings are rated against this threat model, not against generic container security checklists.

Finding	Severity	Description
PA-01	LOW	Missing `CLONE_NEWIPC` in container namespace flags (Linux-only; defense-in-depth)
PA-02	LOW	Seccomp filter blocks only 2 syscalls — default-allow policy (appropriate for trusted-agent model)
PA-03	LOW	No cgroup resource limits on containers (trusted workloads; DoS from own agents is self-inflicted)
PA-04–15	INFO	Positive: no setuid, PR_SET_NO_NEW_PRIVS enforced, capabilities dropped, environment cleared, fork+execvp, pivot_root with overlayfs, –privileged properly guarded

---

14. Windows Deployment Analysis

Agent: windows-deployment | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 1 Medium, 3 Low, 9 Info

Windows Security Highlights

- `CreateProcess` -based execution (no `cmd.exe /c` with user input)
- Metacharacter validation on shell arguments
- SHA256-verified self-updates
- User-scoped installation (no admin/UAC required)
- Proper argument quoting in Windows command construction

Finding	Severity	Description
WIN-01	MEDIUM	`scorpiox-bash.c` writes temp `.bat` files in `%TEMP%` for cmd.exe execution (TOCTOU risk in shared temp; mitigated by user-scoped %TEMP%)
WIN-02	LOW	`scorpiox-wsl.exe` SHA256 verification bypass possible on parse error (dead code path — `check_for_update()` never called)
WIN-03	LOW	DLL search order — `SetDllDirectory` used but `libsx.dll` loaded from `%LOCALAPPDATA%\scorpiox` (trusted path)
		`scorpiox-multiplexer` uses named pipes

WIN-04	LOW	without explicit ACL (defaults to creator-only access on Windows)
WIN-05–13	INFO	Positive: CreateProcess usage, SHA256 update verification, user-scoped PATH, no admin required, proper quoting

---

15. Telemetry and Tracking

Agent: telemetry-tracking | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 0 Medium, 1 Low, 4 Info

Verdict: No Tracking by Default

Both telemetry systems (`USAGE_TRACKING` and `EMIT_SESSION_TRACKING`) are **disabled** (`0`) in all configuration tiers: compiled defaults, embedded config, and shipped config file. No network telemetry occurs out of the box.

Mandatory Data Collection: NONE — All telemetry is fully opt-in.

System	Default	Data Collected (when enabled)
`USAGE_TRACKING`	OFF	Token counts, model name, session UUID, hostname, OS (no conversation content)
`EMIT_SESSION_TRACKING`	OFF	Full conversation messages, tool calls/results (privacy-sensitive; requires explicit opt-in)

Finding	Severity	Description
TEL-01	LOW	`USAGE_TRACKING` guard logic uses negative check — if config value is NULL/missing, tracking would proceed (defaults to `"0"` so safe in practice)
TEL-02–05	INFO	Positive: both systems disabled by default, gating at both caller and binary level, no mandatory collection, opt-in architecture

---

16. Install Script & Distribution Security

Agent: install-script | **Risk:** MEDIUM | **Findings:** 0 Critical, 0 High, 3 Medium, 7 Low, 7 Info

Distribution Security Architecture

- All downloads use HTTPS with HSTS on `dist.scorpiox.net`
- SHA256 checksum verification enforced by default on all platforms
- Per-commit build provenance via `build-info.json`
- Windows installer uses user-scoped PATH (no admin required)

Finding	Severity	Description
IS-01	MEDIUM	Same-origin checksums — binary and SHA256 hash served from same server; no independent code signing (GPG, Sigstore)
IS-02	MEDIUM	`index.txt` branch control file has no integrity protection (server compromise could redirect to malicious branch)

IS-03	MEDIUM	`verify_sha256_update()` parse-error bypass in `scorpiox-wsl.c` (dead code path — function never called)
IS-04	LOW	`--no-verify` flag allows checksum bypass (Linux/macOS only)
IS-05	LOW	Non-atomic installation — partial state on interruption
IS-06	LOW	No HSTS on `get.scorpiox.net` installer endpoint
IS-07	LOW	macOS binary uses ad-hoc codesign (no Developer ID)
IS-08	LOW	No rollback mechanism on installation failure
IS-09	LOW	`curl|bash` pattern susceptible to partial download execution (Linux/macOS)
IS-10	LOW	No reproducible build process
IS-11–17	INFO	Positive: SHA256 enforced by default, HTTPS with HSTS on dist server, per-commit provenance, user-scoped Windows install

---

17. WPF Launcher (scorpioxcode.exe)

Agent: wpf-launcher | **Risk:** LOW | **Findings:** 0 Critical, 0 High, 1 Medium, 5 Low, 16 Info

WPF Security Highlights

- Zero NuGet package dependencies (all .NET Framework BCL)
- P/Invoke bindings with proper free-after-copy pattern
- SHA256-verified dependency downloads
- Static callback delegate pinning (prevents GC collection)
- No embedded browser (eliminates XSS-class risks)
- Deterministic build enabled

Finding	Severity	Description
WPF-01	MEDIUM	`CharSet.Ansi` marshalling corrupts non-ASCII input to UTF-8 native API (functional bug with minor security implications)
WPF-02	LOW	Unbounded null-terminator scan in `PtrToStringUtf8` (trusted first-party library contract)
WPF-03	LOW	Install script URL lacks explicit `https://` scheme
WPF-04	LOW	Binary and checksum from same origin (no code signing)
WPF-05	LOW	Remote telemetry sends machine name and file paths (minor PII)
WPF-06	LOW	`AutoUpdater` SHA256 verification — same-origin checksum pattern
WPF-07–22	INFO	Positive: zero NuGet deps, proper P/Invoke safety, SHA256 downloads, no unsafe code,

		deterministic builds

18. Consolidated Risk Matrix

Medium-Severity Findings

#	Area	Finding	Severity	Windows	Recommendation
1	Credentials	Hardcoded OAuth client IDs (public, no secret)	MEDIUM	✅	Make configurable for rotation
2	Credentials	Hardcoded infrastructure IP in WASM config	MEDIUM	✅	Use DNS hostname consistently
3	File I/O	API bodies logged in plaintext (0600 perms)	MEDIUM	✅	Add config to disable body logging; log rotation
4	File I/O	Path traversal double-encoding bypass	MEDIUM	✅	Use `realpath()` validation consistently
5	File I/O	Predictable `/tmp/sxmux` socket path	MEDIUM	❌	Always append UID to path
6	Memory	`sx_api_init()` allocation leak on error	MEDIUM	✅	Remove redundant early return (line 108)
7	Memory	`scorpiox-unshare` stack leak on pipe fail	MEDIUM	❌	Add `free(stack)` before error return
8	Command Inj.	`scorpiox-init` descriptor field injection	MEDIUM	❌	Use fork+execvp or validate with allowlist
9	Command Inj.	`scorpiox-init` INIT_TOOLS env injection	MEDIUM	❌	Add `is_safe_shell_arg()` validation
10	Install	Same-origin checksums (no code signing)	MEDIUM	✅	Implement GPG or Sigstore signing
11	Install	`index.txt` branch control unprotected	MEDIUM	✅	Add integrity verification for index file
12	Install	SHA256 parse-error bypass (dead code)	MEDIUM	✅	Fix parse-error return handling or remove dead code
13	Windows	Temp `.bat` file TOCTOU in scorpiox-bash	MEDIUM	✅	Use unique temp filenames with mkstemp pattern
14	WPF	CharSet.Ansi corrupts non-ASCII to UTF-8 API	MEDIUM	✅	Use manual UTF-8 marshalling for string parameters

Low-Severity Findings (Summary by Area)

Area	Count	Key Themes
Supply Chain	1	Version tracking manifest
Build Provenance	2	Bridge Makefile hardening, PIE
Network Endpoints	3	WASM hardcoded IP, DNS bind, registry TLS
TLS Security	3	SX_TLS_VERIFY toggle, opportunistic STARTTLS, session tickets
Credentials	3	Internal service URLs, config placeholders, SMTP constants
File I/O	5	Log rotation, temp cleanup, error-path handling
Buffer Safety	3	Single sprintf, yamux overflow guard, IMAP truncation
Memory Safety	2	Terminal resize, header realloc
Command Injection	3	Config URL interpolation, Python path, residual sx_system_safe
Privilege & Access	3	CLONE_NEWIPC, seccomp depth, cgroup limits
Windows Deployment	3	Dead-code SHA256 bypass, DLL search, named pipe ACL
Telemetry	1	Usage tracking guard logic
Install Script	7	–no-verify, non-atomic install, HSTS, macOS codesign, curl|bash, rollback, reproducibility
WPF Launcher	5	PtrToStringUtf8 bounds, install URL scheme, same-origin checksums, telemetry PII, auto-updater

---

19. Conclusion & Recommendations

Overall Assessment

SCORPIOX CODE achieves a **LOW** overall risk rating with **zero Critical and zero High findings** across 155 total observations from 14 audit agents. This represents a **significant improvement** from the previous audit (225 findings including 2 Critical and 6 High). The commit under review specifically addresses Windows command injection vulnerabilities, demonstrating active security remediation.

Top Recommendations (Priority Order)

1. **Implement independent code signing** (GPG or Sigstore) for release artifacts to establish a trust anchor independent of the distribution server (IS-01)
2. **Fix** `CharSet.Ansi` **marshalling** in the WPF launcher to prevent non-ASCII input corruption (WPF-02)
3. **Add log rotation and configurable body logging** to prevent indefinite sensitive conversation persistence (FIO-01)
4. **Use** `realpath()` **validation** consistently for all path traversal checks (FIO-02)
5. **Remove dead code** in `scorpiox-wsl.c` `check_for_update()` or fix the SHA256 parse-error handling (IS-03)
6. **Add input validation** for `scorpiox-init` descriptor fields and `INIT_TOOLS` environment variable (CJ-01, CJ-02, Linux-only)

7. **Fix the `sx_api_init()` memory leak** by removing the redundant early return at line 108 (MEM-01)
8. **Expand seccomp blocklist** with additional commonly-blocked syscalls for defense-in-depth (PA-02, Linux-only)

Strengths

- Exceptional buffer safety discipline (3,464:1 safe-to-unsafe ratio)
- Centralized TLS configuration with fail-closed defaults
- Zero third-party runtime dependencies in Windows distribution
- Both telemetry systems disabled by default with no mandatory collection
- Active security remediation visible in commit history
- Clean supply chain — only 2 vendored libraries, both current LTS/stable releases
- Comprehensive security hardening flags in build system

Appendix: Audit Methodology

Agents Deployed

#	Agent	Scope
1	supply-chain	Package managers, vendored libraries, dependency analysis
2	build-provenance	Build system, compiler/linker flags, code generation
3	network-endpoints	Hardcoded URLs, IPs, domains, ports, binding addresses
4	tls-security	Certificate verification, protocol versions, cipher suites
5	credential-hardcode	API keys, passwords, tokens, secrets in source
6	file-io	File operations, temp files, permissions, data at rest
7	buffer-safety	sprintf/strcpy/strcat usage, bounds checking, format strings
8	memory-safety	malloc/realloc NULL checks, use-after-free, double-free, leaks
9	command-injection	system(), popen(), exec*() with untrusted input
10	privilege-access	setuid, root requirements, sandboxing, namespaces
11	windows-deployment	Windows binary enumeration, platform-specific risks
12	telemetry-tracking	Phone-home behavior, analytics, data collection defaults
13	install-script	Installation scripts, update mechanism, distribution integrity
14	wpf-launcher	WPF desktop application security (P/Invoke, downloads, telemetry)

Analysis Approach

- **Static analysis** of 224 non-vendor C/H source files (148,135 lines)
- **Configuration review** of embedded defaults, shipped config files, and build system
- **Threat model-aware rating**: Findings rated against the actual deployment context (trusted first-party agents on private network), not against generic container/server security checklists
- **Platform-scoped findings**: Each finding tagged with platform applicability (Windows/Linux/macOS/All)
- **Delta analysis**: Comparison with previous audit at commit `6851bc0` to track remediation progress

Severity Definitions

Severity	Definition
Critical	Remotely exploitable without authentication; immediate data breach risk
High	Exploitable with minimal prerequisites; significant security impact
Medium	Exploitable with specific prerequisites (local access, config modification, filesystem control); moderate impact
Low	Defense-in-depth hardening; theoretical risk with minimal practical exploitability
Info	Positive security observation or architectural note