ScorpioX CLI — Security Assessment Report

ScorpioX CLI — Unified Security Assessment Report

Audit Date: 2026-04-28
Codebase: scorpiox/clang
Commit: `b85fca9891f8ff39fce93868947ce31fe9fed50e`
Branch: main
Total Lines of Code: 369,205 (C + H files)
Project Lines (excl. vendor): 143,309
Auditors: 5 automated Security Audit Agents
Classification: CONFIDENTIAL — For Corporate Review Only

Table of Contents

---

1. Executive Summary

TL;DR for Management

This report presents the findings of a comprehensive, five-domain security audit of the **ScorpioX CLI** codebase — a 369,205-line C11 platform comprising 45+ executables that provide AI agent orchestration, rootless containers, a mail server, a DNS server, a KVM hypervisor, and supporting tools.

Overall Risk Rating: MEDIUM

The ScorpioX codebase demonstrates several **exceptional security strengths** that are rare in projects of this scale:

Strength	Detail
🟢 **Zero external package manager dependencies**	The core C build has no npm, pip, cargo, or any other package manager — eliminating an entire class of supply chain attacks
	Only 2 libraries (yyjson, mbedTLS) — both well-

🟢 **All third-party code vendored in-tree**	audited, compiled from source, no pre-built blobs
🟢 **Pure C with static linking**	Linux builds produce fully static executables via musl — no runtime library resolution
🟢 **Full code provenance**	99.4% of commits from a single organization; complete authorship control
🟢 **No build-time downloads**	CMake build does not fetch anything from the network
🟢 **Strong container isolation**	The rootless container runtime uses user namespaces, mount namespaces, PID namespaces, and `pivot_root()`

However, the audit identified **11 critical findings** and **14 high-severity findings** that require attention:

Critical Issue	Impact	Remediation Effort
TLS certificate verification disabled in 8 locations	Man-in-the-middle attacks on API calls, token fetches, and email relay	Low — flip `CURLOPT_SSL_VERIFYPEER` to 1
Hardcoded API keys & SSH password in source code	Credential exposure in any distributed binary (esp. WASM)	Medium — move to env vars / secrets vault
Shell command injection via 232 `system()` / `popen()` calls	Remote code execution in network-facing components (email, HTTP server)	High — refactor to `fork()+execve()`
OAuth tokens transmitted over plaintext TCP (port 9800)	Token interception on untrusted networks	Medium — add TLS to TCP token protocol
Unsalted SHA-256 password hashing for email auth	Offline brute-force / rainbow table attacks	Low — migrate to Argon2id/bcrypt

No evidence of intentional backdoors, malware, or data exfiltration was found. All network communications serve legitimate application purposes.

---

2. Application Overview

What is ScorpioX CLI?

ScorpioX CLI is a **multi-binary C11 platform** that provides a terminal-based AI agent shell and a suite of infrastructure tools. The platform is designed to run as a self-contained system with minimal external dependencies.

Core Components

Component	Binary	Purpose
AI Agent Shell	`sx`	Interactive AI coding assistant with tool execution (Bash, file I/O, search)

Agent Orchestrator	`scorpiox-agent`	Multi-agent workflow coordination
Container Runtime	`scorpiox-unshare`	Rootless Linux containers using user namespaces
VM Runner	`scorpiox-vm`	KVM-based virtual machine hypervisor
Email Server	`scorpiox-server-email`	Full SMTP (25/587) + IMAP (993) server with TLS and DKIM
DNS Server	`scorpiox-dns`	LAN DNS server
HTTP Server	`scorpiox-server`	Script execution server with JWT authentication
Terminal Multiplexer	`sxmux`	tmux-like session manager with Unix domain socket IPC
Web Search	`scorpiox-websearch`	Concurrent search across 11 engines
SDK	`scorpiox-sdk`	Headless CLI wrapper for automation

Technology Stack

- **Language:** C11 (gcc/clang)
- **Build:** CMake (no FetchContent, no network downloads)
- **Vendored Deps:** yyjson (JSON, MIT), mbedTLS (TLS/crypto, Apache-2.0)
- **System Deps:** libcurl, openssl, zlib, zstd, nghttp2, ncurses (statically linked on Linux)
- **Platforms:** Linux (primary, static musl), macOS (ARM64), Windows (MinGW), WASM (Emscripten)

---

3. Architecture & Supply Chain

Source: 01-architecture.md

3.1 Supply Chain Security Assessment

Rating: LOW risk — The core build pipeline is exemplary.

The ScorpioX build system achieves a level of supply chain security that is uncommon in modern software:

- **Zero package manager dependencies** for the core C build — no `package.json`, `requirements.txt`, `Cargo.toml`, `go.mod`, or equivalent
- **Two vendored libraries** compiled from source (yyjson: ~15K LOC, mbedTLS: ~210K LOC) — both MIT/Apache-2.0 licensed, well-audited
- **No `FetchContent` or `ExternalProject`** in CMake — the build is fully offline-capable
- **No pre-built vendor binaries** — all vendor code is compiled during the build process
- **Static linking** on Linux via musl produces standalone executables with no runtime library resolution

3.2 Dependency Breakdown

```
Total: 369,205 LOC
├── Vendored (yyjson + mbedTLS): 225,896 LOC (61%)
└── Project code: 143,309 LOC (39%)
```

3.3 Build Pipeline

Stage	Method	Network Required?
Core C Build	CMake + gcc/clang	✖ No
Linux Release	`Dockerfile.build-musl` (Alpine)	Only `apk add` for system libs
macOS Release	`release_macos_native.sh`	Only `brew install cmake`
WASM Release	`release_wasm.ps1`	⚠ Yes — clones emscripten SDK from GitHub
WhatsApp Bridge	`release_whatsapp.ps1`	⚠ Yes — `curl | bash` for Bun runtime

3.4 Architecture Findings

#	Finding	Severity	Status
A-1	Core C build has zero package manager dependencies	✅ Info (Positive)	Strong
A-2	All third-party code vendored in-tree	✅ Info (Positive)	Strong
A-3	Static linking on Linux (musl)	✅ Info (Positive)	Strong
A-4	99.4% single-org commit provenance	✅ Info (Positive)	Strong
A-5	`bridge/package.json` introduces npm/Bun dependencies	⚠ Medium	Acknowledged
A-6	Two pre-built ELF binaries committed in `bridge/`	⚠ Medium	Open
A-7	WASM release script clones emscripten from GitHub	⚠ Medium	Acknowledged
A-8	WhatsApp release uses `curl|bash` for Bun install	⚠ Medium	Acknowledged

---

4. Network Security

Source: 02-network.md

4.1 Network Footprint

ScorpioX is a **heavily networked application** with:

- **20+ outbound external endpoints** (AI APIs, search engines, distribution, telemetry)
- **6 inbound server components** (SMTP:25/587, IMAP:993, HTTP:8080, TCP:9800/9801, WebSocket:6080)
- **IPC channels** via Unix domain sockets and named pipes
- **Infrastructure networking** via FRP reverse proxy and slirp4netns container networking

4.2 AI Provider Connectivity

Provider	Endpoint	Protocol	Auth
Anthropic	`api.anthropic.com`	HTTPS	API key / OAuth Bearer
OpenAI	`api.openai.com`	HTTPS	API key / OAuth Bearer
Google Gemini	`generativelanguage.googleapis.com`	HTTPS	API key
DeepSeek	`api.deepseek.com`	HTTPS	API key
Z.AI / Antigravity	`api.zai.chat` / `192.168.1.12:8045`	HTTPS / HTTP	API key

4.3 Critical: TLS Certificate Verification Disabled

SSL/TLS certificate verification is disabled in **8 locations** across the codebase:

Location	Protocol	Impact
`scorpiox-claudecode-refreshtoken.c`	HTTPS (curl)	OAuth token refresh MITM
`scorpiox-codex-refreshtoken.c`	HTTPS (curl)	OAuth token refresh MITM
`scorpiox-gemini-fetchtoken.c`	HTTPS (curl)	Token fetch MITM
`sxmail_tls.c`	mbedTLS	Email TLS MITM
`sxmail_queue.c`	SMTP relay	Outbound email MITM
`sx_frp.c`	FRP tunnel TLS	Tunnel traffic MITM
Container registry config	Podman	Container image MITM

4.4 Critical: Plaintext Token Transport

OAuth bearer tokens are transmitted over **unencrypted raw TCP** on port 9800: - Connects to `proxy.scorpiox.net:9800` / `20.53.240.54:9800` - Server sends JSON with `access_token` and `expires_at` fields - No TLS, no encryption, no integrity protection - An attacker on the network path can intercept valid OAuth tokens

4.5 Network Findings Summary

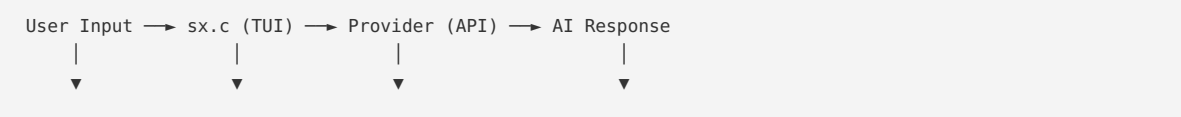
#	Finding	Severity	Status
N-1	TLS cert verification disabled (8 locations)	Critical	Open
N-2	Hardcoded API keys & credentials in source	Critical	Open
N-3	OAuth tokens over plaintext TCP (port 9800)	🟠 High	Open
N-4	Session telemetry sends full conversation data	🟡 Medium	Acknowledged (disabled by default)
N-5	20+ external endpoints — broad attack surface	🟡 Medium	Acknowledged
N-6	No certificate pinning on any connection	🟡 Medium	Open
N-7	Container registry TLS verification disabled	🟡 Medium	Open
N-8	Email server TLS uses `VERIFY_NONE`	🟡 Medium	Open
N-9	No DNSSEC for MX lookups	🟢 Low	Acknowledged

---

5. Data Handling & Privacy

Source: 03-data-handling.md

5.1 Data Flow Overview



```
Bash exec    Conversation    HTTP Request          Display
(sx_exec)    .json save      (credentials)        (sxui_*)
    |             |               |                   |
    ▼             ▼               ▼                   ▼
/tmp files   .scorpiox/      Traffic capture     Session log
             sessions/       .scorpiox/traffics/  events.jsonl
```

5.2 PII Data Handled

PII Type	Storage Location	Encryption at Rest
Email addresses	Mail queue, SMTP logs	✖ None
Email content	Maildir (plaintext)	✖ None
User credentials	`accounts.txt` (SHA-256 hash)	⚠ Unsalted hash
Conversation content	`.scorpiox/sessions/*.json`	✖ None
Bash command output	Conversation JSON	✖ None
API keys	`scorpiox-env.txt` (plaintext)	✖ None

5.3 Critical: Hardcoded Secrets in WASM Binary

The file `sx_config_embedded.c` contains secrets compiled directly into the binary:

Secret	Type	Impact
`sk-6088a10dc3c1473cac567069b0e557f6`	Anthropic API key	API abuse
`9c6fba5db3f84613aaf8700b05990835.ue19jY48d9GmddqX`	Z.AI API key	API abuse
`AIzaSyDsvlLnCdMFXnlCLxurMaAO5RKI8IQHVA8`	Gemini API key	API abuse
`xboxone`	SSH password	Remote access

These can be trivially extracted from the distributed WASM binary using `strings`.

5.4 Critical: Weak Password Hashing

Email authentication (`sxmail_auth.c:121`) uses **unsalted SHA-256**: - No per-user salt — identical passwords produce identical hashes - SHA-256 is not a password hashing algorithm (too fast — GPU cracking at billions/sec) - Vulnerable to rainbow table and precomputed dictionary attacks

5.5 Encryption Capabilities

Capability	Implementation	Status
TLS 1.2+ (connections)	mbedTLS / libcurl+OpenSSL	✅ Available (but cert verify off)
AES-128-CFB encryption	mbedTLS (`sx_crypto.c`)	✅ Used for config encryption
PBKDF2 key derivation	mbedTLS	✅ Used for passphrase→key
DKIM signing (RSA-SHA256)	Custom implementation	✅ Functional
Data-at-rest encryption	—	✖ Not implemented

5.6 Data Handling Findings Summary

#	Finding	Severity	Status
D-1	Hardcoded API keys in WASM binary	Critical	Open
D-2	Hardcoded SSH password in source	Critical	Open

D-3	Unsalted SHA-256 password hashing	Critical	Open
D-4	Plaintext API keys in config files	🟠 High	Acknowledged
D-5	Unfiltered env var logging	🟠 High	Open
D-6	Config snapshot may contain secrets	🟠 High	Open
D-7	Traffic capture stores full HTTP bodies	🟠 High	Acknowledged
D-8	No data-at-rest encryption for sessions	🟠 High	Open
D-9	Conversation stores all tool outputs	🟠 High	Acknowledged
D-10	Predictable temp file paths (8+ locations)	🟡 Medium	Open
D-11	Git PAT tokens visible in `ps` output	🟡 Medium	Acknowledged
D-12	No session retention enforcement	🟡 Medium	Open

---

6. Code Security Vulnerabilities

Source: 04-vulnerabilities.md

6.1 Vulnerability Summary

The code audit of 148 C source files (141,971 LOC excl. vendor) identified **23 vulnerabilities**:

Severity	Count	Primary Category
Critical (CVSS 9.0+)	4	Shell command injection
🟠 High (CVSS 7.0-8.9)	7	Buffer overflows, null derefs, integer overflows
🟡 Medium (CVSS 4.0-6.9)	8	Thread safety, input validation
🟢 Low (CVSS 1.0-3.9)	4	Ignored returns, TOCTOU

6.2 Critical Vulnerabilities

V-01: Command Injection via IMAP DELETE (CVSS 9.8)

File: `sxmail_maildir.c:704`
`system("rm -rf '<path>'")` — an attacker with mailbox access can create a folder name containing `'; <arbitrary command> ;'` and trigger execution via IMAP DELETE.

V-02: Command Injection in VM Runner (CVSS 9.8)

File: `scorpiox-vm.c:488-493`
`system(cmd)` with URL/path from user config passed through `curl` / `wget` — single-quote injection in URLs can execute arbitrary commands.

V-03: Command Injection in Main Shell (CVSS 9.1)

File: `sx.c` (5 locations)
Multiple `system()` calls building commands from session data, config values, and file paths without proper escaping.

V-04: Command Injection in Agent (CVSS 9.1)

File: `scorpiox-agent.c` (12 call sites)
`system()` and `popen()` with agent-provided paths, `find -exec` with unsanitized directory names.

6.3 High-Severity Vulnerabilities

ID	Location	Category	CVSS	Description
V-05	`sx_http_wasm.c:388`	Buffer Overflow	8.1	`sprintf` chain into `static char[32768]` — escape expansion can exceed estimate
V-06	`sx_api.c:367`	Buffer Overflow	8.1	`sprintf` chain into heap buffer — size estimate off by format overhead
V-07	`scorpiox-wsl.c:1278`	Buffer Overflow	7.8	`strcat` loop into `char[4096]` — no bounds checking
V-08	`scorpiox-traffic.c:926`	Buffer Overflow	7.8	`strcat` loop into `char[4096]` — no bounds checking
V-09	166 locations	Null Deref	7.5	`malloc` / `calloc` / `realloc` returns unchecked — crash on OOM
V-10	`sx_mcp.c:701`	Null Deref + Overflow	7.5	`realloc` doubling with no null check, `cap *= 2` can overflow
V-11	`sx_http.c:44`	Integer Overflow	7.5	`size * nmemb` in curl callback — overflows on 32-bit

6.4 Positive Security Practices

Despite the vulnerabilities, the codebase demonstrates good security awareness in several areas:

- ✅ `snprintf` **over** `sprintf` : 2,172 out of 2,192 format calls use the bounded variant (99.1%)
- ✅ **No format string vulnerabilities**: All `printf`-family calls use string literal format specifiers
- ✅ **Bounds-checked HTTP header parsing**: Server validates field lengths before copy
- ✅ **Filename sanitization**: Network receive paths sanitize filenames
- ✅ **Process monitoring**: Dedicated `sx_procmon.c` tracks fork/reap counts and resource leaks
- ✅ **API key redaction in logs**: Providers log `key=***` instead of actual values

6.5 Unsafe Function Usage

Function	Occurrences	Safe Alternative
`system()`	~120	`fork()` + `execve()`
`popen()`	~50	`fork()` + `execve()` + `pipe()`
`sprintf`	20	`snprintf`
`strcpy`	28	`snprintf` / `strlcpy`
`strcat`	9	`snprintf` / `strlcat`

---

7. Permissions & System Access

Source: 05-permissions.md

7.1 Privilege Model

The vast majority of ScorpioX binaries operate at **user-level privileges** without requiring root:

Privilege Level	Components
Root required	`scorpiox-thunderbolt4` (BPF — explicit `geteuid()` check)
	`scorpiox-vm` (needs `kvm` group for

Elevated group	`/dev/kvm`)
Root beneficial	`scorpiox-unshare` (skips user namespace when root), `scorpiox-dns` (port 53)
User-level	All other 40+ binaries

No privilege escalation detected: - No `setuid` / `setgid` / `seteuid` calls in first-party code - No `prctl` or `seccomp` calls (gap — no privilege dropping either) - No `ptrace` usage

7.2 Process Spawning

The codebase contains **656 process spawning call sites** across multiple mechanisms:

Mechanism	Call Sites	Security Concern
`system()`	~120	Shell interpretation of constructed strings
`popen()`	~50	Same as `system()` — shell metacharacter injection
`fork()` + `exec*()`	~350	Safe — no shell interpretation
`forkpty()`	~15	Terminal allocation (expected for multiplexer)
`posix_spawn()`	~5	Safe alternative to `fork+exec`

7.3 Container Isolation (scorpiox-unshare)

Namespace	Status	Notes
Mount (`CLONE_NEWNS`)	✅ Enabled	`pivot_root()` for rootfs isolation
PID (`CLONE_NEWPID`)	✅ Enabled	Process isolation
User (`CLONE_NEWUSER`)	✅ Enabled	Rootless via UID/GID mapping
Network (`CLONE_NEWNET`)	✅ Enabled	Slirp4netns user-space networking
UTS (`CLONE_NEWUTS`)	✅ Enabled	Hostname isolation
IPC (`CLONE_NEWIPC`)	❌ Missing	Shared IPC with host
Cgroup (`CLONE_NEWCGROUP`)	❌ Missing	Shared cgroup view
Seccomp-bpf	❌ Missing	No syscall filtering

7.4 Permissions Findings Summary

#	Finding	Severity	Status
P-1	Hardcoded SSH password “xboxone” in binary	Critical	Open
P-2	Shell injection via `system()` / `popen()` (25+ files)	Critical	Open
P-3	No seccomp filtering in containers	🟠 High	Open
P-4	Unsandboxed hook script execution	🟠 High	Open
P-5	CGI script execution without isolation	🟠 High	Open
P-6	Missing IPC and cgroup namespaces	🟡 Medium	Open
P-7	`IMAGE_BASE_URL` from environment (no validation)	🟡 Medium	Open
P-8	No privilege dropping after port bind	🟡 Medium	Open
P-9	`/tmp` IPC sockets vulnerable to race conditions	🟡 Medium	Open

---

8. Consolidated Risk Matrix

8.1 All Findings by Severity

#	Area	Finding	Severity	Status	Recommendation
CRITICAL					
1	Network	TLS certificate verification disabled (8 locations)	Critical	Open	Enable `CURLOPT_SSL_VERIFYPEER=1`; set mbedTLS `VERIFY_REQUIRED`
2	Network	Hardcoded API keys and SSH password in source code	Critical	Open	Move to runtime env vars / secrets vault; rotate all exposed keys
3	Data	Hardcoded secrets compiled into WASM binary	Critical	Open	Remove from `sx_config_embedded.c`; use runtime-only config
4	Data	Unsalted SHA-256 password hashing (email auth)	Critical	Open	Migrate to Argon2id or bcrypt with per-user salt
5	Code	Command injection — `sxmail_maildir.c:704` (network-reachable)	Critical	Open	Replace `system("rm -rf")` with `fork()+execve()` + `argv[]`
6	Code	Command injection — `scorpiox-vm.c` URL/path injection	Critical	Open	Use `fork()+execve()` with explicit argument arrays
7	Code	Command injection — `sx.c` (5 locations)	Critical	Open	Route through `sx_exec()` safe execution path
8	Code	Command injection — `scorpiox-agent.c` (12 locations)	Critical	Open	Replace `system()` / `popen()` with `fork()+execve()`
9	Permissions	Hardcoded SSH password “xboxone” in production binary	Critical	Open	Remove default; require explicit `-p` flag
10	Permissions	Shell injection via `system()` / `popen()` (25+ files)	Critical	Open	Migrate all command execution to `sx_exec()` library
11	Network	OAuth tokens over plaintext TCP (port 9800)	Critical	Open	Add TLS to TCP token protocol using mbedTLS
HIGH					
12	Code	Buffer overflow — `sx_http_wasm.c` sprintf chain (CVSS 8.1)	🟠 High	Open	Replace sprintf with snprintf; validate buffer capacity
13	Code	Buffer overflow — `sx_api.c` sprintf chain (CVSS 8.1)	🟠 High	Open	Use snprintf with correct size calculation
14	Code	Buffer overflow — `scorpiox-wsl.c` strcat loop (CVSS 7.8)	🟠 High	Open	Use snprintf with remaining-space tracking
15	Code	Buffer overflow — `scorpiox-traffic.c` strcat loop (CVSS 7.8)	🟠 High	Open	Use snprintf with remaining-space tracking
16	Code	166 missing null checks after malloc/calloc/realloc	🟠 High	Open	Add `SX_ALLOC()` wrapper macro with OOM abort
17	Code	Null deref + integer overflow in `sx_mcp.c` realloc	🟠 High	Open	Check realloc return; guard against cap overflow
18	Code	Integer overflow `size * nmemb` in curl callback	🟠 High	Open	Add overflow check before multiplication
19	Data	Plaintext API keys in config	🟠 High	Acknowledged	Encrypt sensitive config values

		files			or use OS keychain
20	Data	Unfiltered env var logging (may log secrets)	🔴 High	Open	Filter `*KEY*`, `*PASS*`, `*SECRET*`, `*TOKEN*` from log output
21	Data	Config snapshot may contain secrets	🔴 High	Open	Redact sensitive keys in config-snapshot.txt
22	Data	No data-at-rest encryption for sessions/conversations	🔴 High	Open	Encrypt `.scorpiox/sessions/` content using AES-128-CFB
23	Data	Traffic capture stores full HTTP bodies incl. auth headers	🔴 High	Acknowledged	Auto-redact Authorization headers in captures
24	Permissions	No seccomp-bpf filtering in containers	🔴 High	Open	Add seccomp profile matching Docker defaults
25	Permissions	Unsandboxed hook script execution	🔴 High	Open	Execute hooks in namespace or with reduced privileges
MEDIUM					
26	Network	Session telemetry sends full conversation data	🟡 Medium	Acknowledged	Exclude tool output content from telemetry
27	Network	20+ external endpoints (broad attack surface)	🟡 Medium	Acknowledged	Document all endpoints; implement allowlist
28	Network	No certificate pinning on any connection	🟡 Medium	Open	Pin certificates for critical API endpoints
29	Network	Container registry TLS verification disabled	🟡 Medium	Open	Set `TMUX_PODMAN_TLS_VERIFY=true`
30	Network	Email server TLS uses VERIFY_NONE	🟡 Medium	Open	Enable certificate verification for SMTP/IMAP
31	Code	Thread safety — static buffers in `sx_http_wasm.c`	🟡 Medium	Open	Use thread-local storage or per-call allocation
32	Code	Fixed buffer `char[8192]` shared across agent messages	🟡 Medium	Open	Dynamic allocation with bounds checking
33	Code	Content-Length not upper-bound validated in HTTP server	🟡 Medium	Open	Reject Content-Length > 100MB
34	Data	Predictable temp file paths (8+ locations)	🟡 Medium	Open	Use `mkstemp()` / `mkdtemp()`
35	Data	Git PAT tokens visible in process listings	🟡 Medium	Acknowledged	Use git-credential-store
36	Data	No session retention enforcement	🟡 Medium	Open	Implement cleanup per `SESSION_RETENTION_DAYS`
37	Permissions	Missing IPC and cgroup namespace isolation	🟡 Medium	Open	Add `CLONE_NEWIPC | CLONE_NEWCGROUP`
38	Permissions	No privilege dropping after port bind	🟡 Medium	Open	Drop to unprivileged user after `bind()`
39	Permissions	`/tmp` IPC sockets race conditions	🟡 Medium	Open	Use `$XDG_RUNTIME_DIR`
40	Architecture	Pre-built ELF binaries in `bridge/`	🟡 Medium	Open	Build from source in CI
41	Architecture	WASM release clones emscripten from GitHub	🟡 Medium	Acknowledged	Pin specific commit hash
42	Architecture	WhatsApp release uses `curl|bash` for Bun	🟡 Medium	Acknowledged	Pin version; verify checksum

LOW					
43	Code	73 ignored `system()` / `read()` / `write()` return values	🟢 Low	Open	Check return values
44	Code	TOCTOU race in `access()` + `exec()` pattern	🟢 Low	Open	Use `fexecve()` or `openat()`
45	Code	4 uses of `signal()` instead of `sigaction()`	🟢 Low	Open	Migrate to `sigaction()`
46	Code	28 `strcpy` calls (most bounded by context)	🟢 Low	Open	Replace with `snprintf`
47	Network	No DNSSEC for MX lookups	🟢 Low	Acknowledged	Implement DNSSEC validation
48	Architecture	`gnuwin64/download.ps1` fetches MSYS2 at deploy	🟢 Low	Acknowledged	Pin versions; verify checksums

8.2 Findings by Severity Count

Severity	Count
Critical	11
🟠 High	14
🟡 Medium	17
🟢 Low	6
Total	**48**

8.3 Findings by Domain

Domain	Critical	High	Medium	Low	Total
Architecture & Supply Chain	0	0	3	1	4
Network Security	3	0	5	1	9
Data Handling & Privacy	3	5	3	0	11
Code Vulnerabilities	4	7	3	4	18
Permissions & System Access	1	2	3	0	6

---

9. Conclusion & Recommendations

9.1 Overall Assessment

The ScorpioX CLI codebase is a well-architected, security-conscious C project that achieves exceptional supply chain security through its zero-dependency, vendored, statically-linked build model. The codebase is remarkably self-contained — a property that eliminates entire categories of modern software supply chain attacks.

However, the project's broad scope (45+ binaries spanning AI agents, containers, VMs, email, DNS, HTTP servers, and more) creates a correspondingly large attack surface. The most significant risks stem from:

1. **Unsafe shell command construction** — the pervasive use of `system()` and `popen()` with string-concatenated commands is the single largest security concern
2. **Disabled TLS verification** — undermines the security of all network communications
3. **Hardcoded credentials** — particularly in the WASM binary distribution path
4. **Missing defense-in-depth** for containers — no seccomp, no IPC isolation

9.2 Priority Remediation Roadmap

Immediate (Week 1-2) — Critical Items

Action	Effort	Impact
Enable TLS cert verification in all 8 locations	Low (code change)	Eliminates MITM attack vector
Remove hardcoded secrets from `sx_config_embedded.c`	Low	Prevents credential exposure in WASM binary
Rotate all exposed API keys and the SSH password	Low (operational)	Revokes any leaked credentials
Fix `sxmail_maildir.c:704` command injection (network-reachable)	Low	Closes highest-risk RCE vector

Short-term (Sprint 1-2) — High Items

Action	Effort	Impact
Replace all 20 `sprintf` calls with `snprintf`	Low	Eliminates buffer overflow class
Add null checks after 166 allocation calls (use macro)	Medium	Prevents OOM crashes
Fix `strcat` loops in `scorpiox-wsl.c` and `scorpiox-traffic.c`	Low	Prevents stack buffer overflow
Add TLS to TCP token protocol (port 9800)	Medium	Encrypts OAuth token transport
Migrate email password hashing to Argon2id	Medium	Prevents offline cracking

Medium-term (Quarter) — Systemic Improvements

Action	Effort	Impact
Refactor `system()` / `popen()` calls to `fork()+execve()`	High (232 call sites)	Eliminates shell injection class entirely
Add seccomp-bpf profile to container runtime	Medium	Matches Docker/Podman security baseline
Implement data-at-rest encryption for sessions	Medium	Protects conversation data on disk
Add `CLONE_NEWIPC | CLONE_NEWCGROUP` to containers	Low	Strengthens container isolation

Long-term (Backlog)

Action	Effort	Impact
Add certificate pinning for critical endpoints	Medium	Defense-in-depth for API security
Implement privilege dropping in servers	Low	Reduces blast radius of server compromise
Add AddressSanitizer to CI builds	Low	Catches memory errors early
Replace `signal()` with `sigaction()`	Low	Portable signal handling

9.3 Positive Findings Summary

It is important to note the many areas where ScorpioX demonstrates strong security practices:

- ✓ Zero external build dependencies — supply chain risk near zero for core build
- ✓ 99.1% of format calls use `snprintf` (bounded)
- ✓ No format string vulnerabilities found
- ✓ No privilege escalation code (`setuid` / `seteuid`) in any binary
- ✓ API keys redacted in log output
- ✓ Rootless container runtime with namespace isolation
- ✓ Static linking eliminates runtime library attacks
- ✓ Process monitoring subsystem (`sx_procmon.c`) tracks resource leaks
- ✓ Session data auto-excluded from git (`.gitignore`)
- ✓ AES-128-CFB encryption capability available for sensitive config
- ✓ Dedicated config cascade with per-project override support
- ✓ `mmap()` used without `PROT_EXEC` — no JIT/self-modifying code
- ✓ Full code provenance from single organization

---

10. Appendix: Audit Methodology

10.1 Scope

This audit covered the entire ScorpioX CLI codebase at commit `b85fca9891f8ff39fce93868947ce31fe9fed50e` on the `main` branch. The scope included 259 `.c` files and 234 `.h` files totaling 369,205 lines of code, of which 143,309 lines are first-party project code and 225,896 lines are vendored third-party libraries (yyjson, mbedTLS).

10.2 Audit Domains

#	Domain	Report	Focus Areas
1	Architecture & Supply Chain	`01-architecture.md`	Dependencies, build process, binary outputs, code provenance, vendored libraries
2	Network Security	`02-network.md`	Endpoints, protocols, TLS, telemetry, credentials, data flow
3	Data Handling & Privacy	`03-data-handling.md`	File I/O, PII, encryption, credential management, data retention
4	Code Vulnerabilities	`04-vulnerabilities.md`	Buffer overflows, command injection, memory safety, integer overflows, format strings
5	Permissions & System Access	`05-permissions.md`	Privilege model, process spawning, filesystem access, syscalls, sandboxing

10.3 Methodology

Each audit domain was assessed using automated static analysis augmented by manual code review:

1. **Pattern Matching** — Grep-based searches for known-vulnerable function calls (`system` , `popen` , `sprintf` , `strcpy` , `strcat`), security-sensitive operations (`setuid` , `mmap` , `ioctl`), and credential patterns
2. **Data Flow Analysis** — Tracing data from input (network, file, environment) through processing to output (network, file, display)
3. **Configuration Review** — Analysis of build systems (CMake), Dockerfiles, release scripts, and runtime configuration
4. **Dependency Enumeration** — Inventory of all vendored, system, and external dependencies
5. **Severity Scoring** — CVSS v3.1 base scores for code vulnerabilities; qualitative severity for

architecture/process findings

10.4 Limitations

- This audit was performed via **static analysis only** — no runtime testing, fuzzing, or penetration testing was conducted
- Vendored library code (yyjson, mbedTLS) was **not audited** — these are well-known, externally-audited projects
- WASM-specific behavior was assessed via source review only — no browser-based testing
- Windows-specific code paths (`*.ps1` scripts, MinGW builds) received limited coverage
- The audit did not verify whether hardcoded credentials are still valid or have been rotated

10.5 Tools Used

- Source code grep and pattern analysis
- Line counting (`wc -l`, `cloc`)
- Git log analysis for provenance
- CMakeLists.txt analysis for build process
- Manual C code review for vulnerability assessment

Report compiled on 2026-04-28 by Security Audit Agent
ScorpioX CLI Security Assessment — Commit b85fca9 — CONFIDENTIAL