Security Policy - Nyx AI

This document covers how to report security issues, what we treat as in-scope, and how we respond.

The companion document `THREAT-MODEL.md` lays out what Nyx AI contains, what it doesn't, and the deliberate trust boundaries the architecture enforces. Read that for the conceptual picture; read this for the process.

Reporting a vulnerability

Please report security problems privately — do not post details publicly (forums, social media, public issue trackers) before coordinated disclosure:

- **Email:** `customerservice@nyxlimited.com` — put `SECURITY` in the subject line so the report is triaged ahead of general support.

When reporting, please include:

1. A short description of the issue
2. Steps to reproduce, or the smallest proof-of-concept that demonstrates the problem
3. The affected Nyx AI version (find it in About > version pill)
4. Your platform (Windows 10/11, build number is helpful)
5. Whether you've shared this report with anyone else

We aim to acknowledge receipt within **48 hours** on weekdays.

In scope

We will investigate and prioritise reports about:

Sandbox escape

- Anything that lets a model-issued action (a `shell_run` call, a `file_write`, etc.) escape the active execution boundary: AppContainer Locked execution (shown in the app's Safety settings as the Sandbox levels — "Sandbox + internet" / "Sandbox · no internet") where active, or the Job Object fallback/resource tier where Locked execution is unavailable or not selected
- Filesystem writes outside the workspace root that bypass `workspace-broker.validatePath`
- Known-dangerous command families or obfuscated/indirect command forms reaching execution despite the command-risk screen. Please report the smallest safe reproduction privately rather than publishing a reusable payload in an issue.
- **MCP server escaping the enabled per-MCP containment** when `sandboxMcp` is enabled: process spawn outside the selected backend, memory/process cap bypass in the Standard tier, or lifecycle cleanup not firing on Nyx AI exit. The documented Standard-tier degradation path is excluded: if Windows refuses the Job assignment, Nyx AI keeps the trusted extension running with normal user authority and visibly marks containment degraded in its card and audit log.

- **Hook escaping the enabled per-hook containment** when `sandboxHooks` is enabled: same containment promises as MCP
- **Low-integrity drop not actually taking effect** when `lowIntegrity` (1.2.32) is enabled: the integrity drop reporting success but the child process running at medium integrity
- Approval-policy bypasses: any path that lets a tool run when it should have been hard-blocked, shown to the user, natively confirmed, or explicitly auto-allowed only by the user's selected mode/rule

Audit log integrity

- Tampering with the encrypted `sandbox-audit.enc` log without the hash-chain verifier detecting it
- Decryption attacks against the AES-256-GCM scheme
- DPAPI-key recovery without same-user-account credentials (this in-scope class applies where the master key is protected by DPAPI/ `safeStorage` ; the documented path-derived fallback provides tamper evidence, not equivalent key confidentiality)
- Skipping audit log entries for actions that should be recorded

Secret handling

- API keys leaking to disk in plaintext
- API keys being transmitted anywhere we don't explicitly document
- Keys remaining recoverable after the user clicks "Delete all data"
- DPAPI key exposure via Electron's safeStorage

Update / supply chain

- Bypassing the Authenticode signature check on release installer builds (public release builds are signed as NYX Limited)
- Bypassing the update's signature / SHA-512 verification before an installer can run
- The update-check IPC exfiltrating user data
- Compromised npm dependencies in our build that ship to users

Privacy

- Telemetry, crash reports, or analytics being sent without explicit user consent
- Session data, audit logs, or memory contents leaking to third parties
- Local IP / system info being included in any outbound request

Renderer integrity

- XSS / arbitrary code execution in the renderer process
- IPC channel abuse from a compromised renderer
- CSP bypass in the preview iframe leading to escape

Remote content and malicious files

- User-triggered web fetch/search, document-image fetch, preview, or generated-content flows bypassing the documented URL/network guards
- A browser-backed render starting with Chromium's own OS sandbox disabled without a main-owned native user confirmation for that exact executable and current app session, or reusing that grant for another executable/session
- Malicious or misleading files causing model-issued tools to bypass the workspace boundary, approval policy, sanitizer, or rendering limits

Out of scope

These aren't accepted as vulnerabilities (we may still appreciate the report as a usability or robustness note, but it won't be treated as a security incident):

- **User-installed extensions.** MCP servers, plugins, and hooks are trusted user extensions. By default they run with the user's normal permissions and are trusted at install/configuration time, not individually re-approved for every tool call. If `sandboxMcp` or `sandboxHooks` is enabled, their child process may be routed through the selected execution backend for resource/lifecycle containment, but Nyx AI deliberately does not apply the model shell command/path blocklist to them because the user installed or authored them. A malicious MCP/plugin/hook remains the user's responsibility; see `THREAT-MODEL.md` section 3.
- **Bring-your-own-key API exposure.** If the user puts their OpenAI/Anthropic key into Nyx AI, that key is in process memory while Nyx AI is running. A malicious local process running as the same user could read it. This is the standard local-app threat boundary; we don't claim memory protection beyond DPAPI at rest.
- **Social engineering of the user via the model.** If a model produces output that tricks the user into running something, that's a model-prompt issue, not a Nyx AI vulnerability. The approval queue is the safety net; if the user clicks Approve, that's a user decision.
- **Compromise of the user's Windows account.** If an attacker has the user's account credentials, they can do anything the user can. Nyx AI doesn't and can't protect against that.
- **Issues solely in Electron, Chromium, or Node itself.** Report the upstream flaw to that project. If it has a Nyx AI-specific impact, reachable product path, or missing mitigation, report that impact to us as well; we assess app-level mitigations and adopt patched releases.
- **Denial of service via prompt injection.** A model that gets stuck in a loop or burns user tokens isn't a security incident; it's a cost / UX issue. We still want to know, just not via this channel.
- **SmartScreen warnings on unsigned builds.** Public release builds are Authenticode-signed as NYX Limited, but local/dev builds are unsigned and may trigger SmartScreen; reputation also takes time to build for a newly signed publisher. That's expected, not a vulnerability.

Response process

When you report a valid issue:

1. **Within 48 hours (weekdays):** we acknowledge receipt.
2. **Within 7 days:** we send an initial assessment (confirmed / declined / need more info) and a target fix timeline.
3. **Fix development:** depends on severity and complexity:
   - Critical (active exploit, sandbox escape, audit-log forgery): we aim for a fix within 7 days
   - High (sandbox bypass with non-trivial preconditions, secret leak): 14-30 days
   - Medium / Low: next regular release
4. **Disclosure:** we coordinate with you on timing. Default is 90-day responsible disclosure: we'll publish the advisory and fix on the 90th day after your report (or sooner if we ship the fix earlier). We credit reporters who want to be credited.

We do not currently run a paid bug bounty. We're a small project; if you find something serious we'll absolutely credit you in the advisory and the release notes.

Cryptography and hardening claims

This is what we actually do, so reports can be measured against it:

- **Always-on model-command floor:** brokered file/path tools enforce the configured workspace and any additional roots the user explicitly allows. Model-issued shell commands still go through built-in hard command refusals. Model-issued shell and Python both go through risk classification, secret-environment scrubbing, and the applicable approval policy. When execution isolation is off, however, a new raw shell process starts in the workspace while Python is staged and starts in a Nyx AI temporary directory. The child process's own paths are not screened, and both use the user's normal filesystem and network permissions. Standard and AppContainer add the execution layers described below. These controls are defense in depth; approval is the final user-control boundary for ordinary model-issued commands. User-started Build and enabled trusted-workspace verification rely on the prior authorization and prompt-free rules described below. This floor is not an OS filesystem jail for arbitrary user-installed MCP, hook, plugin, LSP, or project-server child processes.
- **Execution isolation, Automatic mode:** the default resolver uses the Windows AppContainer backend where the capability check succeeds. If that check fails, or secure journal initialization fails before any AppContainer ACL transaction starts, Automatic mode falls back to the Standard Win32 Job Object tier and surfaces the reduced boundary in the UI. A partial ACL transaction, cleanup-repair state, or explicit Locked choice remains fail-closed instead of becoming a downgrade. A user who explicitly selects AppContainer/Locked execution gets fail-closed behaviour instead of a silent downgrade. During startup, before a backend is active, screened commands may run on the always-on floor without an OS jail; the active-tier indicator is the authority.

- **Locked execution (AppContainer):** shell/Python work runs in a Windows AppContainer profile scoped to the workspace. Writable access covers the workspace, Nyx AI-managed scratch/dependency directories and explicit allowed roots; narrower read/execute access is also granted where required for interpreters, toolchains and caches. Grant intent is recorded in a local encrypted recovery journal before permissions change. A revocation that cannot be verified stays recorded for retry and is not counted as completed cleanup or confirmed data erasure. The OS enforces filesystem access to granted workspace paths, and the no-internet profile drops outbound network capability. Nyx AI also denies writes from the contained process to protected workspace control files where applicable. This is the strongest execution tier, but it remains risk reduction rather than a guarantee against every possible OS, Electron, or implementation flaw.
- **Standard execution (Job Object):** Win32 Job Objects provide process lifetime/resource controls, including memory/process limits and child cleanup, plus command/path text screening before launch. This tier is **not** a kernel filesystem jail and is **not** a complete network jail. The dangerous-command and command-obfuscation screens are pinned by automated tests. Their exact detection patterns are retained privately.
- **Optional containment settings:**
  - `sandboxMcp` : routes MCP server spawns through the selected active backend where supported. Automatic may use AppContainer or fall back to the Standard Job Object tier; an explicitly selected Sandbox level refuses the child if AppContainer is unavailable. Command/path blocklists are not applied.
  - `sandboxHooks` : the same backend/fail-closed rules for lifecycle hooks in `.nyx/hooks.json` , with `cmd.exe /d /s /c` wrapping so shell pipelines work.
  - `lowIntegrity` : in the Standard tier, lowers the wrapper process's Windows integrity level and verifies the resulting token before spawning. If setup or verification fails, the child is not started. The child inherits Low integrity, so the kernel blocks writes to medium-integrity objects independently of the path policy. This setting is not used for AppContainer, which is the stronger boundary.
- **Audit log:** AES-256-GCM authenticated encryption with a unique IV per record. A SHA-256 hash chain across records detects insertion, deletion, or modification. When Electron `safeStorage` is available, the master key is 32 random bytes wrapped by the OS keychain (DPAPI on Windows). If OS wrapping is unavailable or fails, Nyx AI derives a key with scrypt from the local `userData` path. That fallback preserves the log's tamper evidence but is not equivalent key confidentiality: another local user who can read that path may be able to recompute it.
- **Secrets at rest:** API keys entered in the dedicated key fields are `safeStorage` -encrypted in `settings.json` when real OS encryption is available. On Windows this normally uses DPAPI. If it is unavailable, a new key remains in memory for that session and is not written to disk; existing encrypted blobs are retained byte-for-byte until they can be decrypted again. Non-secret preferences are plaintext JSON. A username/password embedded in an OpenAI, Anthropic, Ollama Cloud, or cloud-embeddings base URL is rejected at save — the URL, and any API key

submitted with it, is not stored — and Ollama Local URLs likewise reject embedded credentials. Nyx AI is Windows-first, and Windows is the supported release target.

- **No telemetry egress.** Every known network call is enumerated in the Network Egress section of `THREAT-MODEL.md`. Most calls happen only when the user acts in that moment (chat message, web fetch tool, MCP marketplace fetch). A small number of background calls can occur only for features that are enabled, configured, or on by default and can be turned off — the update check (on by default; a single request carrying the app version, while the host necessarily sees the IP address, request time, and standard connection/ request metadata; opt-out in Settings → Advanced → Updates), scheduled tasks, automatic chat titles and background model helpers (which prefer local models but can follow configured role routing to a cloud provider), Ollama Cloud key checks after current Terms acceptance (at startup and after relevant configuration changes), provider model-catalog discovery (`GET /v1/models` on startup when a key is saved), local autocomplete, local/cloud embeddings, first-use browser downloads, language-server prewarming/downloads when a compatible trusted workspace opens, enabled MCP server connection/reconnection and component bootstrap behaviour, and approved plugin-repository clones. None of these calls are telemetry, analytics, crash reporting, or install-ID tracking.
- **No telemetry.** No crash reporter, no analytics, no install-ID transmission. The on-by-default update check is not telemetry — it sends the app version to fetch a version file; the host also sees the IP address, request time, and standard connection/request metadata, but the request carries no analytics or generated identifiers and can be turned off in Settings → Advanced → Updates.

If you find any claim above that's not actually true, that's a security report; please file it.

Verifying your download (integrity)

Official Windows **release** builds are **Authenticode-signed** as NYX Limited — check the signature via right-click the `.exe` → Properties → Digital Signatures, or `Get-AuthenticodeSignature`. (Local dev builds are unsigned.) Either way, also verify your download against the published checksums so a swapped or tampered build doesn't go unnoticed:

1. Each release ships a `SHA256SUMS.txt` alongside the `.exe` files.
2. Compute the hash of your download and compare it to that file:

   ```
   Get-FileHash .\Nyx-AI-<version>-portable.exe -Algorithm SHA256
   ```

   The printed hash must exactly match the line for that filename in `SHA256SUMS.txt`. If it doesn't, **do not run it**; re-download from the official source and report a mismatch via the channels above.

`SHA256SUMS.txt` is generated automatically as the last step of every release build. Because public release builds are Authenticode-signed (via `npm run package:signed`), the Windows publisher field / SmartScreen reputation is the primary integrity signal and these checksums are a secondary cross-check.

Severity rubric

We use a simplified CVSS-inspired rubric:

Severity	Criteria
Critical	Sandbox escape, audit-log forgery, remote code execution, secret exfiltration
High	Sandbox bypass with non-trivial preconditions, privilege escalation within Nyx AI, persistent denial-of-service of the user's workspace
Medium	Misleading security indicator, partial bypass that requires user interaction, key-handling flaw that needs local code-execution to exploit
Low	Information disclosure of non-sensitive data, error messages revealing internal paths, UI states that misrepresent actual security state

Coordinated disclosure timing

Default: **90 days** from confirmed receipt. We'll publish:

- A clear writeup in the release notes and on our website
- A CVE if applicable
- The fix release version
- Credit to the reporter (unless they prefer to remain anonymous)

If we can't fix within 90 days for legitimate reasons (e.g., the issue is in an upstream Chromium release we depend on), we'll explain why and propose a new timeline. We won't sit on a critical vulnerability indefinitely; at some point publishing forces the fix.

If you find a critical issue and we don't respond within 7 days, escalate by emailing again with `SECURITY ESCALATION` in the subject line, or via any contact route published on nyxai.uk.

What we ask from reporters

- **Don't exfiltrate data** beyond what's necessary to demonstrate the issue. Showing you *can* read a file isn't the same as reading every file.
- **Don't perform DoS testing** against the user's machine.
- **Don't share the issue publicly** until we've had a reasonable chance to fix it. We commit to coordinating disclosure with you in good faith.
- **Be patient.** This is a small project. We'll respond as fast as we reasonably can.

Thank you for helping make Nyx AI safer.

Licence carve-out for security research

Nyx AI is licensed under the **Nyx AI End User Licence Agreement (EULA) v2.9** (see `LICENSE` at the repo root). Apart from mandatory statutory rights that cannot be excluded (such as the lawful-user rights preserved under sections 50A, 50B and 50BA of the Copyright, Designs and Patents Act 1988), that licence forbids reverse engineering, modification, distribution, and derivative works in general. **Section 4 of the licence carves out an explicit exception for good-faith security research** acting under this disclosure policy.

Specifically, while researching a vulnerability you may:

- Reverse engineer the binaries to the extent necessary to identify and demonstrate the issue
- Modify your local copy solely to the extent necessary to confirm the behaviour of a finding
- Share minimum-necessary code excerpts, traces, or proof-of-concept material privately with us, and after coordinated disclosure in a published advisory

You may NOT, under the carve-out:

- Exfiltrate or share other people's data
- Test against systems not under your authorisation
- Publish exploits before coordinated disclosure timing
- Use or distribute a modified copy for any purpose other than confirming and demonstrating the finding
- Repackage the source code under another licence

If you have any doubt about whether a research activity is covered, ask us at `customerservice@nyxlimited.com` before proceeding; we'd much rather say yes in advance than dispute it after the fact.

Current main-process enforcement notes

Unattended scheduled runs cannot send to external web tools after reading workspace data. New workspace roots require the native picker; renderer restore data is not authority and Nyx AI's own user-data tree is excluded independently. Repository-controlled commands outside Locked execution require the documented native trust decision; one Allow then covers other repository-controlled commands in that workspace for the rest of the app session (memory only, never persisted, cleared on restart), while commands classified high or critical keep their own per-command confirmation. Unattended work that cannot obtain a required decision is refused. Ordinary model-issued high/critical commands keep their native gate. A user-started Build flow and enabled Auto-verify may run the screened command classes disclosed in the Terms without a second per-command prompt; Auto-verify does so in a workspace the user trusted, or in any workspace while a Sandbox execution level is live; Auto-verify refuses wrapper commands classified blocked, high-risk or critical, but project scripts invoked by an allowed wrapper run as trusted repository code and their bodies are not semantically screened. Locked filesystem and network authority are not widened by any of these controls.

Published by **NYX LIMITED**, a company registered in England and Wales (company number 17261427), registered office 70 Clarkehouse Road, Sheffield, England, S10 2LJ. ICO registration **ZC167877**. Contact: customerservice@nyxlimited.com.