Ollama Local Privacy & Persistence

Evidence-led research and implementation checklist

Project baseline v1.0 | Miigwech AI Solutions| 3 November 2026

Executive assessment

Ollama has multiple distinct persistence surfaces. The inspected source directly demonstrates desktop chat storage in SQLite; attachment BLOBs; thinking text and tool interactions; a separate chat-associated image directory; interactive CLI history and a temporary replacement file; optional full-request debug dumps; and external-editor prompt files. Explicitly saving a CLI session also sends its conversation messages into model creation. These are separate mechanisms, not alternative names for one chat-history store.
[web:39][web:42][web:43][web:36][web:68]

This report is a public-source implementation review, not a completed forensic examination of a shipped binary. Source evidence proves the identified code paths exist in the inspected artifacts. It does not prove that every installed Ollama version executes them, that all possible write sites have been found, or that operating-system and third-party copies can be erased. No local Ollama installation was examined and no deletion experiments were performed.

The product goal should be: prevent supported future persistence where possible; remove selected application-managed interaction artifacts; verify observable results; and report coverage gaps without promising universal forensic erasure.

1. Method, scope, and evidence grades

The review used Ollama repository source, official documentation, SQLite documentation, and upstream issue/PR reports. Desktop, CLI, server, and external integrations are treated separately. Issue reports are corroborating leads, not independent experimental proof.

Evidence grades used throughout:

```
A — Direct implementation evidence: a retrieved write, read, deletion, or path-construction operation.
B — Official documentation or underlying SQLite behaviour.
C — Upstream issue or proposed feature; not proof of a shipped implementation.
U — Investigation target; not established as an Ollama interaction store by this review.
```

Version boundary: app/store/database.go was inspected at commit f9f4af6ce09223fde24949a5b3ccff061ca0abf4, schema version 19. Other retrieved source pages use mutable main. A retrieval date is not a release identifier. Before shipping rules, resolve each main-based finding to a commit and test the actual supported binary. [web:39]

Do not reuse GitHub's indexed publication date for a source file as the date a feature was introduced. This review does not establish the latest released version or map every finding to a production release.

2. Evidence dossiers: what is stored and how we know

E01 — Desktop database locations [A]

Source: internal/onboarding/app_state.go, AppDatabasePath(); app/store/store.go, defaultDBPath and ensureDB().

The path constructor returns these locations:

```
Windows: %LOCALAPPDATA%\Ollama\db.sqlite
macOS: ~/Library/Application Support/Ollama/db.sqlite
Other platforms: empty path in the inspected constructor.
```

The store takes this default or its DBPath override, creates the parent directory, and passes the path to newDatabase(). This connects the path definition to the actual database-opening operation. The desktop store

source has a windows || darwin build constraint. Do not infer an official Linux desktop database from these findings. [web:45][web:42]

Cleanup consequence: discover the resolved database, including custom stores and all relevant user identities. Default paths are discovery seeds, not a complete inventory.

E02 — Conversation content is actually written into SQLite [A]

Source: commit-pinned app/store/database.go; init(), saveChat(), appendMessage(), updateLastMessage(), updateChatBrowserState().

The schema defines chats, messages, tool_calls, attachments, users, and settings. More importantly, saveChat() inserts/upserts chat metadata, loops through chat.Messages, calls insertMessage(), and inserts tool calls. updateLastMessage() explicitly updates content, thinking, model_name, timestamps and tool_result; it also replaces attachments and tool calls. updateChatBrowserState() writes serialized browser state. This is write-path evidence, not merely inference from table names. [web:39]

Content inventory:

- chats: id, title, created_at, browser_state.
- messages: role, content, thinking, model_name, timestamps, thinking timestamps, serialized tool_result.
- tool_calls: function_name, function_arguments, function_result, type and relationships.
- attachments: filename and data BLOB, linked to a message.
- users: cached name, email, plan and timestamp; personal metadata rather than a transcript.
- settings: device identifier, working directory, model selection and feature/integration preferences; not all are interaction content. [web:39]

The saveChat() comment explicitly describes preserving the last persisted full tool state through browser_state. Its complete serialized contents require a browser/tool audit; this report does not equate it with an entire browser profile. [web:39]

Cleanup consequence: cover every sensitive table and field. Clearing messages.content alone is incomplete. Preserve or reset users/settings according to an independently selected identity/configuration mode.

E03 — Historical orphan records [A]

Source: database.go, cleanupOrphanedData().

The implementation independently deletes tool_calls whose message is missing, attachments whose message is missing, and messages whose chat is missing. A source comment associates this with a previous foreign-key enforcement bug. This proves orphan handling exists; it does not prove that a particular installation still contains orphaned content. [web:39]

Cleanup consequence: inspect sensitive tables independently. Do not rely solely on UI-visible chats or joins from chats. Avoid starting the application just to inspect it if startup migrations would alter the evidence.

E04 — SQLite WAL and deletion remnants [A/B]

Source: database.go, newDatabase(), Close(), deleteChat(); SQLite temporary-file and deletion documentation.

The database connection requests _foreign_keys=on, _journal_mode=WAL, _busy_timeout=5000 and _txlock=immediate. Close() and deleteChat() attempt PRAGMA wal_checkpoint(TRUNCATE). The inspected deleteChat() performs DELETE FROM chats WHERE id = ?; it is not an explicit secure-erasure routine. [web:39]

Assess the complete database family:

- db.sqlite: live records and potentially deleted remnants in reusable pages.
- db.sqlite-wal: changed database pages; potentially sensitive content.
- db.sqlite-shm: WAL index/coordination structure, not a second transcript database.
- db.sqlite-journal: conditional rollback-journal candidate, not the normal companion under WAL. [web:64]

SQLite documents that ordinary deletion usually marks space reusable rather than erasing its contents. VACUUM and secure_delete address aspects of deleted-content handling, but neither should be advertised as erasing every copy in backups or underlying storage. SQLite can also create temporary files for maintenance and query processing. [web:50][web:51][web:64]

Upstream issue #16544 reports recoverable deleted chats in a Windows database. Treat this as a report requiring reproduction, not a measured result from this project. [web:19]

Cleanup consequence: quiesce clients, inspect actual settings, use a tested SQLite maintenance procedure, verify integrity and sidecars, and do not delete a live WAL independently.

E05 — Separate chat-associated image storage [A, path unresolved]

Source: app/store/store.go, DeleteChat().

Decisive code:

```
chatImgDir := filepath.Join(s.ImgDir(), id)
if err := os.RemoveAll(chatImgDir); err != nil { ... }
```

The accompanying comment says associated images are deleted. Failure is logged but does not fail overall chat deletion. This establishes an independently managed image directory and a possible partial-cleanup outcome. It does not establish that every attachment is duplicated there. [web:42]

The image.go search result identifies an ImgDir implementation, but full retrieval failed. Therefore the absolute image path and complete image-write lifecycle remain unresolved here. Do not substitute a guessed path. [web:66]

Cleanup consequence: resolve ImgDir() for each supported release, inspect chat subdirectories, handle orphans, and verify removal separately from database deletion.

E06 — CLI input history and replacement file [A]

Source: readline/history.go, Init(), Add(), Save().

Init() joins os.UserHomeDir(), .ollama and history. Save() derives tmpFile := h.Filename + ".tmp", writes every buffered history line, and renames it over the final file. The inspected history object defaults to autosave enabled, history enabled, and a 100-entry buffer limit. [web:43]

Paths:

```
<home>/.ollama/history
<home>/.ollama/history.tmp
```

This is readline input history, not necessarily a complete assistant transcript. The source does not justify treating the 100-entry limit as erasure of prior filesystem copies. interactive.go invokes HistoryDisable() when NoHistory() is true; it also allows /set history and /set nohistory within a session. [web:43][web:68]

Cleanup consequence: remove both files after closing active sessions, apply CLI no-history prevention, and verify that sessions do not recreate content. Shell history remains a different surface.

E07 — Full-request debug dumps [A, conditional]

Source: server/inference_request_log.go, initRequestLogging(), middleware(), log().

The logger is enabled only when envconfig.DebugLogRequests() is true. It creates a directory using os.MkdirTemp("", "ollama-request-logs-*"). Middleware reads the incoming request body and restores it for request handling. log() writes the body with os.WriteFile(..., body, 0o600) and creates a replay script. [web:36]

Artifact patterns:

```
<process temp>/ollama-request-logs-*/<timestamp>_<route>_body.json

<process temp>/ollama-request-logs-*/<timestamp>_<route>_request.sh
```

The script references the JSON file. It is not itself necessarily a second full transcript. The inspected file logs requests, not response bodies; route coverage requires checking the middleware registration in the exact release. The directory is announced in ordinary server logs. No cleanup routine appears in this inspected file; this alone does not prove no other component ever removes it. [web:36]

Cleanup consequence: disable OLLAMA_DEBUG_LOG_REQUESTS independently of OLLAMA_DEBUG, discover the actual server temp root, remove existing dump families, and inspect past-run directories.

E08 — External-editor prompt files [A, newly confirmed]

Source: cmd/interactive.go, editInExternalEditor().

Decisive operations:

```
os.CreateTemp("", "ollama-prompt-*.txt")
defer os.Remove(tmpFile.Name())
tmpFile.WriteString(content)
cmd := exec.Command(args[0], args[1:]...)
os.ReadFile(tmpFile.Name())
```

The CLI creates a prompt text file, writes initial prompt content when present, launches the selected editor on that path, then reads the resulting prompt. Normal control flow schedules file removal. Abnormal termination can bypass normal deferred cleanup; editor-generated recovery files require separate testing. [web:68]

Artifact pattern: <CLI process temp>/ollama-prompt-*.txt.

Cleanup consequence: include leftover prompt files and runtime tests for editor swap/autosave/recovery artifacts. This was an investigation-only item in the earlier checklist; the primary prompt-file write is now directly confirmed.

E09 — Explicitly saved sessions can enter model creation [A/B, conditional]

Source: cmd/interactive.go, /save handler and NewCreateRequest(); Modelfile documentation.

The CLI advertises /save <model> as saving the current session. The handler calls NewCreateRequest() then client.Create(). NewCreateRequest() clones opts.LoadedMessages, appends opts.Messages, and assigns them to req.Messages. It also includes a nonempty opts.System. This directly proves conversation data can be submitted into model creation through the save workflow. Modelfile documentation separately supports MESSAGE history and SYSTEM content. [web:68][web:69]

This report has not traced the server-side model-creation write into every exact blob/manifest format. Do not describe ordinary inference as rewriting model weights, and do not assume all downloaded models contain the user's chat. The specific conditional concern is user-created session/model metadata containing supplied messages or system text.

Cleanup consequence: add a separate saved-session/custom-model audit. Resolve actual model storage, inspect user-created message/system metadata, and offer selective removal. Trace server create/write and blob lifecycle before claiming verified removal from model storage. Model weights should not be automatically deleted merely to clear chat history.

E10 — Ordinary logs [B/C; transcript content not universally proven]

Official troubleshooting documentation identifies macOS ~/.ollama/logs/server.log, Windows %LOCALAPPDATA%\Ollama\server.log and rotations, Linux service journal output, container logs, and manually captured server output. A Windows issue also shows app.log alongside server/database artifacts. [web:33][web:13]

These are confirmed logging surfaces, not proof that every normal log contains complete prompts/responses. Full request-body persistence has separate direct evidence under E07. Log sensitivity and rotation naming must be tested per release. Temporary executable-extraction directories must not be automatically classified as conversation stores. [web:36][web:33]

E11 — Identity/configuration artifacts [A/B; distinct scope]

The database stores cached users and settings. The desktop source migrates device ID and first-run state from legacy config.json under the Windows Ollama local-app-data directory or macOS Application Support/Ollama directory. The migration structure only establishes those migrated fields, not the complete historical file format. [web:39][web:42]

The configuration source and FAQ describe ~/.ollama/server.json and cloud-disable configuration. Preserve privacy-hardening settings unless a complete reset is selected. Keys and model storage under .ollama have separate purposes; do not recursively delete the directory as a synonym for chat cleanup. [web:34][web:16]

3. Preventive controls and their limits

The following are settings to inspect, apply through supported configuration channels, restart as needed, and verify in effective behaviour. They do not clean existing artifacts. [web:34][web:16]

- OLLAMA_NO_CLOUD=1: disable Ollama cloud features; not proof of universal offline operation.
- OLLAMA_NOHISTORY=1: disable CLI readline history; not desktop no-storage.
- OLLAMA_DEBUG_LOG_REQUESTS=0: disable explicit request dumps; independently controlled from ordinary debug logging.
- OLLAMA_DEBUG=0: reduce debug logging; ordinary logs remain.
- OLLAMA_HOST=127.0.0.1:11434: configure loopback listening; not a complete local-access access-control boundary.
- OLLAMA_KEEP_ALIVE=0: optional reduction of idle model retention; requests may override it, and unloading is not verified memory sanitization. [web:34][web:16]

Desktop source exposes Expose, Browser, Survey, Agent, Tools, WorkingDir and WebSearchEnabled. Survey is described as permission to share anonymous inference information; Browser is described as allowing exposure to browser windows/CORS. Do not confuse this Browser setting with proof of a browser-profile store. Use version-supported settings mechanisms and assess the desired functionality/privacy trade-off. [web:42]

OLLAMA_ORIGINS is additive in the inspected implementation; do not advertise it as replacing all built-in origins. A proposed OLLAMA_GUI_NOHISTORY feature is not established production support. Even clearing GUI history at startup/shutdown is not equivalent to preventing writes. [web:34][web:18]

4. External and operating-system scope [U]

The following are candidates for controlled testing, not proved Ollama-owned interaction stores in this report: shell history; terminal transcripts; clipboard history/synchronization; webview caches/local storage/IndexedDB; editor recovery files; original attachments; tool working directories and generated files; third-party frontends; proxy request recording; tracing systems; crash/core dumps; swap/pagefile/hibernation; backups/snapshots/synchronization; container volumes and writable layers; VM/WSL disks; and the cleaner's own logs and crash reporting.

Classify each candidate as detected, inspected, cleaned, excluded by user, inaccessible, unsupported or untested. Remote copies and third-party account retention require separate authority and retention analysis. Disabling future cloud use is not deletion of earlier remote interactions.

5. Implementation checklist with evidence traceability

Status fields for the project: Not started / Implemented / Tested / Blocked / Not applicable. Priority: P0 safety/discovery, P1 core privacy coverage, P2 extended surfaces, P3 environmental investigation. Checkboxes are intentionally uncompleted.

A. Scope, discovery and safety

[ ] A01 | P0 | Offer audit, hardening, conversation cleanup, identity reset, full reset, and model cleanup separately. Evidence: E01-E11.

[ ] A02 | P0 | Resolve installed versions, executable identity, install type, process owner and relevant instances. Rule prerequisite.

[ ] A03 | P0 | Resolve actual home, temp, database and model roots; support custom DBPath. Evidence: E01, E06-E09.

[ ] A04 | P0 | Recognize database schemas and refuse unknown destructive migrations. Evidence: E02.

[ ] A05 | P0 | Preview resolved targets; reject unexpected symlinks/reparse targets and scope escapes. Safety requirement.

[ ] A06 | P0 | Obtain explicit consent for deleting identity keys, original files, models, backups and third-party data. Safety requirement.

[ ] A07 | P0 | Quiesce desktop, CLI, server and clients; prevent concurrent repopulation. Evidence: E02, E04, E06.

[ ] A08 | P0 | Report locked, inaccessible or unassessed locations; do not collapse partial success into success.

[ ] A09 | P0 | Avoid automatic sensitive backups, transcript logging and uploads by the cleaner itself.

B. Prevent future persistence/exposure

[ ] B01 | P1 | Apply and verify OLLAMA_NO_CLOUD or supported equivalent. Section 3.

[ ] B02 | P1 | Apply OLLAMA_NOHISTORY; verify CLI sessions and in-session re-enabling behaviour. E06.

[ ] B03 | P1 | Disable OLLAMA_DEBUG_LOG_REQUESTS; verify dump creation stops. E07.

[ ] B04 | P1 | Disable debug logging separately; document remaining normal logging. E10.

[ ] B05 | P1 | Verify loopback-only listener and disable unwanted desktop Expose/Browser settings. Section 3.

[ ] B06 | P1 | Inspect Survey and disable optional sharing when supported. Section 3.

[ ] B07 | P2 | Inspect unnecessary agent/tool/web-search capabilities and working directories. Section 3.

[ ] B08 | P2 | Offer shorter model retention only with an accurate non-erasure explanation. Section 3.

[ ] B09 | P0 | Restart as required, then check effective settings; preserve privacy settings during cleanup.

[ ] B10 | P0 | Never claim CLI no-history prevents desktop writes or that a proposed GUI setting is supported.

C. Desktop content and database family

[ ] C01 | P1 | Locate all default and custom db.sqlite instances. E01.

[ ] C02 | P1 | Clean chat titles, timestamps and browser_state. E02.

[ ] C03 | P1 | Clean message content, thinking, tool_result and associated metadata. E02.

[ ] C04 | P1 | Clean tool-call arguments/results/names according to selected scope. E02.

[ ] C05 | P1 | Clean attachment filenames and BLOB data. E02.

[ ] C06 | P1 | Inspect and remove orphan tool_calls, attachments and messages. E03.

[ ] C07 | P1 | Inventory main database, WAL, SHM and any rollback journal. E04.

[ ] C08 | P1 | Implement tested selective deletion/remnant handling; inspect actual SQLite settings. E04.

[ ] C09 | P0 | Do not independently delete a live WAL; verify maintenance and integrity results. E04.

[ ] C10 | P1 | Full reset: remove complete resolved database family only after closure and user approval. E04.

[ ] C11 | P2 | Resolve ImgDir() per release, clean chat image directories and inspect orphans. E05; currently blocked on path verification.

[ ] C12 | P2 | Verify image removal independently; surface failures hidden by successful chat deletion. E05.

D. CLI, request dumps, editor and saved models

[ ] D01 | P1 | Remove .ollama/history for each relevant identity. E06.

[ ] D02 | P1 | Remove .ollama/history.tmp and verify no active session recreates it. E06.

[ ] D03 | P1 | Discover process temp roots and remove ollama-request-logs-* artifact families. E07.

[ ] D04 | P1 | Include both *_body.json and

_request.sh; inspect logs for recorded paths. E07.[ ] D05 | P1 | Remove leftover ollama-prompt-.txt files within verified scope. E08.

[ ] D06 | P2 | Test editor autosave/swap/recovery; implement editor-specific cleanup only with evidence and consent. E08/U.

[ ] D07 | P1 | Detect user-created saved sessions/models and inspect message/system metadata. E09.

[ ] D08 | P2 | Trace model creation to exact manifests/blobs and test shared-content lifecycle before claiming complete saved-session cleanup. E09; unresolved.

[ ] D09 | P0 | Keep ordinary downloaded models out of automatic conversation deletion; make saved-session removal explicit. E09.

E. Logs, identities and broader scope

[ ] E01 | P2 | Inventory current, rotated and archived app/server logs by deployment. E10.

[ ] E02 | P2 | Identify journal/container/external logging backends and selective-removal limits. E10.

[ ] E03 | P2 | Remove or sanitize selected logs without wiping unrelated system logs. E10.

[ ] E04 | P2 | Offer cached-user/device/legacy-config reset separately. E11.

[ ] E05 | P2 | Offer explicit credential/key reset; preserve required privacy configuration. E11.

[ ] E06 | P3 | Audit shell/terminal/clipboard and third-party client storage. Section 4.

[ ] E07 | P3 | Audit webview/tool/browser outputs and user exports. Section 4.

[ ] E08 | P3 | Assess dumps, swap/hibernation, backups, snapshots and synchronized copies. Section 4.

[ ] E09 | P3 | Assess containers, VM/WSL layers, external tracing and remote retention separately. Section 4.

F. Verification and maintenance

[ ] F01 | P0 | Verify selected artifacts are absent and retained user assets remain intact.

[ ] F02 | P1 | Verify remaining database integrity and expected row counts without recording content.

[ ] F03 | P1 | Recheck temporary files, images, WAL and history after restart.

[ ] F04 | P1 | Run synthetic-canary tests for every supported interface and content type.

[ ] F05 | P1 | Test normal exit, interruption, crash, deletion, restart and custom paths.

[ ] F06 | P1 | Inspect logical records and raw-file remnants; report decoding/search limitations.

[ ] F07 | P1 | Observe filesystem writes and network destinations under the supported privacy preset.

[ ] F08 | P0 | Bind cleanup rules to verified release/schema/commit combinations; retest after upgrades.

[ ] F09 | P0 | Produce a content-free audit report with per-target outcomes and coverage gaps.

6. Reproducible validation protocol

1. Use a fresh VM or dedicated account; record OS, Ollama binary hash/version, configuration and baseline files.
2. Use different synthetic markers for user text, assistant output, system content, thinking output where available, attachment name/body, tool argument/result, browser/tool state, editor prompt and saved-session messages.
3. Exercise desktop, CLI, API, request debugging, editor invocation and explicit /save workflows separately. Do not infer one interface's behaviour from another.
4. Capture filesystem writes, process identities and network destinations. Record actual resolved roots.
5. Inspect SQLite records, sidecars, raw files, image directories, request dumps, editor files and saved-model metadata before deletion.
6. Repeat with normal shutdown, forced termination, failed removals, restart, single-chat deletion and full cleanup.
7. Verify the cleaner does not create a new copy through backup, debugging, telemetry or crash reporting.
8. Publish per-release results: source prediction, observed artifact, operation performed, verification result and limitations.

A plain-text canary search alone is insufficient: content can be encoded, compressed, transformed, serialized or fragmented. A negative search is a bounded observation, not a universal no-remnants proof. No actual canary experiment is claimed by this report.

7. Open questions and release gates

- Resolve image.go / ImgDir(), image write paths and cache lifecycle. Primary chat-associated directory evidence exists; absolute path remains unverified.
- Pin all main-based evidence to commits and map to supported released binaries.
- Trace saved-session model creation through exact on-disk metadata/blob paths and deletion/pruning behaviour.
- Audit webview profile settings, browser/tool state and tool working-directory outputs.
- Audit every log/error path and exact request-logging route coverage.
- Test CLI initialization with NOHISTORY: the inspected flow constructs readline before invoking HistoryDisable(); do not assume the flag guarantees no history-file access or creation without checking the constructor/runtime.
- Assess credentials, shared onboarding records and integration metadata separately from interaction stores.
- Validate OS-level, backup and third-party scopes only for the environments the product explicitly supports.

Suggested acceptance wording: Application-managed artifacts in the reported scope were removed and verified. Listed exclusions, inaccessible locations, external retention and forensic recovery beyond the tested scope remain unverified.

8. Evidence index and reproducibility notes

The source identifiers below match the inline citations. They identify the specific artifacts supporting the findings; mutable pages must be archived/pinned during project validation.

[web:39] Ollama app/store/database.go, commit f9f4af6ce09223fde24949a5b3ccff061ca0abf4. Schema 19; database opening, schema, write paths, orphan cleanup, deletion and checkpoint.

[web:42] Ollama app/store/store.go, retrieved main. Store override, settings, legacy migration, image deletion and API wrappers.

[web:45] Ollama internal/onboarding/app_state.go, retrieved main. AppDatabasePath().

[web:43] Ollama readline/history.go, retrieved main. History path, autosave, temporary replacement file.

[web:36] Ollama server/inference_request_log.go, retrieved main. Conditional request dumps and replay scripts.

[web:68] Ollama cmd/interactive.go, retrieved main. NOHISTORY integration, external editor prompt file, /save and NewCreateRequest message handling.

[web:69] Ollama docs/modelfile.mdx, indexed main documentation. MESSAGE and SYSTEM instructions.

[web:66] Ollama app/store/image.go, indexed search result only. ImgDir lead; full retrieval unsuccessful.

[web:34] Ollama envconfig/config.go, retrieved main in preceding review. Environment controls and cloud configuration.

[web:16] Official Ollama FAQ, preceding review. Local/cloud distinction, environment configuration, model storage and relevant settings.

[web:33] Official Ollama troubleshooting documentation, preceding review. Platform/deployment logging and temporary executable storage.

[web:50] SQLite VACUUM documentation. Deleted-content reuse and database rebuilding.

[web:51] SQLite PRAGMA documentation. secure_delete and related settings.

[web:64] SQLite Temporary Files Used By SQLite. WAL, SHM, journals and temporary maintenance/query files.

[web:19] Ollama issue #16544. Reported deleted-chat recovery; not independently reproduced.

[web:18] Ollama pull request #14326. Proposed GUI history control; not evidence of released support.

[web:13] Ollama issue #11289. Report of Windows SQLite files beside app/server logs.

9. Revision log

v1.0 — 3 November 2026. Consolidated evidence before checklist; explicitly separated direct proof from candidates; added newly confirmed external-editor prompt files and CLI saved-session/model-creation message flow. Retained unresolved image path and server-side saved-model storage tracing as open gates. No binary/runtime testing performed.