Claw Code (Python)

audited.xyz
by zack.eth

Date
Apr 03, 2026 00:41 UTC

Commit
1abd951
69 files, 2,711 lines

Summary

Claw Code (Python) is a CLI-based porting workspace that mirrors the architectural surface of a TypeScript agent harness. The application loads static JSON snapshot data for command and tool inventories, provides a query engine for routing prompts to mirrored command and tool entries, and persists lightweight session state to the local filesystem. It does not expose any network services, web endpoints, or database connections — all interaction occurs through `argparse`-driven CLI subcommands. The codebase is structured around frozen dataclasses, explicit module exports, and a permission model that governs tool enumeration.

The codebase demonstrates generally sound security practices for a local CLI tool. It uses only `json.loads` for deserialization, avoiding pickle and unsafe YAML loaders entirely. There are no subprocess or shell calls, no dynamic code evaluation via `exec` or `eval`, and no imports of cryptographic or network libraries. All snapshot data is loaded from hardcoded, relative paths within the project tree, and the permission system (`ToolPermissionContext`) correctly applies deny-list filtering before tool enumeration, ensuring that restricted tools cannot be surfaced to the query engine.

The overall security posture is appropriate for a development-phase porting workspace, with one low-severity finding identified. A path traversal concern exists in the session persistence layer, where user-supplied session IDs flow into filesystem paths without normalization. While the current CLI-only context limits the practical impact — the local user already has filesystem access — this pattern represents a concrete risk that warrants remediation, particularly as the project evolves toward a networked agent runtime where session identifiers may originate from untrusted sources.

Findings

1 issues identified

1 Low

L-1 LOW Open

Path Traversal in Session Store via Unsanitized Session ID

`</> src/session_store.py:28`

Description

The load_session and save_session functions construct file paths by directly interpolating a session_id string without validating the resolved path stays within the session directory. The CLI exposes this via the load-session subcommand, and QueryEnginePort.from_saved_session accepts arbitrary session_id input. An attacker controlling session_id could read arbitrary .json files or write session data to arbitrary filesystem locations.

Impact

Arbitrary read of .json-suffixed files and limited arbitrary file write on the filesystem. Risk escalates from low (CLI context) to medium/high if these functions are reused in a networked service.

Recommendation

Resolve the constructed path and validate it remains within the target session directory using Path.is_relative_to() before performing any read or write operation.

Fix

The fix resolves the constructed file path to its absolute canonical form and then verifies it remains within the intended session directory using Path.is_relative_to(). This prevents directory traversal sequences like '../' in session IDs from escaping the session storage directory.

`src/session_store.py`

```
- def save_session(session: StoredSession, directory: Path | None = None)→ Path:
-     target_dir = directory or DEFAULT_SESSION_DIR
-     target_dir.mkdir(parents=True, exist_ok=True)
-     path = target_dir / f'{session.session_id}.json'
-     path.write_text(json.dumps(asdict(session), indent=2))
-     return path
-
-
- def load_session(session_id: str, directory: Path | None = None)→ StoredSession:
-     target_dir = directory or DEFAULT_SESSION_DIR
-     data = json.loads((target_dir / f'{session_id}.json').read_text())
-     return StoredSession(
-         session_id=data['session_id'],
-         messages=tuple(data['messages']),
```

```
-         input_tokens=data['input_tokens'],
-         output_tokens=data['output_tokens'],
-     )
+ def save_session(session: StoredSession, directory: Path | None = None)→ Path:
+     target_dir = directory or DEFAULT_SESSION_DIR
+     target_dir.mkdir(parents=True, exist_ok=True)
+     path = (target_dir / f'{session.session_id}.json').resolve()
+     if not path.is_relative_to(target_dir.resolve()):
+         raise ValueError(f'Invalid session_id: path escapes session directory')
+     path.write_text(json.dumps(asdict(session), indent=2))
+     return path
+
+
+ def load_session(session_id: str, directory: Path | None = None)→ StoredSession:
+     target_dir = directory or DEFAULT_SESSION_DIR
+     path = (target_dir / f'{session_id}.json').resolve()
+     if not path.is_relative_to(target_dir.resolve()):
+         raise ValueError(f'Invalid session_id: path escapes session directory')
+     data = json.loads(path.read_text())
+     return StoredSession(
+         session_id=data['session_id'],
+         messages=tuple(data['messages']),
+         input_tokens=data['input_tokens'],
+         output_tokens=data['output_tokens'],
+     )
```

Conclusion

Claw Code (Python) demonstrates solid foundational security practices for a development-stage CLI porting workspace. The codebase avoids all high-risk Python patterns: there is no dynamic code evaluation, no unsafe deserialization, no shell invocation, and no network-facing attack surface. The use of frozen dataclasses, explicit `__all__` exports, and a structured permission model reflects considered design discipline and a security-conscious development approach.

The low-severity path traversal finding (M-1) in the session persistence layer poses a real risk that should be addressed before the project moves beyond local development use. An attacker — or a malicious input source in a future networked context — could craft session identifiers containing directory traversal sequences to read or write files outside the intended session directory. The fix is straightforward: normalizing and validating session ID inputs against a canonical base path before constructing filesystem paths. Addressing this now is inexpensive and prevents the pattern from propagating as the codebase grows.

The application is appropriate for its current purpose as a local development and porting tool. Before any evolution toward a networked agent runtime or multi-user deployment, a follow-up review should evaluate authentication boundaries, input validation at network entry points, and session integrity guarantees. For the current scope and intended use, the codebase is in sound security standing once the identified path traversal pattern is remediated.