ARC XT

A governance layer for AI-assisted coding

ARC XT — Security White Paper

v0.1.13 · Clean Baseline Release · April 5, 2026

Technical deep-dive into the security architecture, threat model, and privacy guarantees of ARC XT.

github.com/habrahgithub/arc-extension · Apache-2.0 License
This document corresponds to ARC XT v0.1.13.

> ***Version 0.1.13*** *· Clean Baseline Release A governance layer for AI-assisted coding in VS Code*

Table of Contents

---

1. Executive Summary

ARC XT is a **local-first governance extension** for VS Code that introduces a tamper-evident audit layer between code generation and version control. It addresses a specific security gap: in AI-assisted development workflows, code is generated faster than it can be reviewed, creating a risk of unvetted changes reaching production.

Security Design Principles

Principle	Implementation
Local-first	All classification and decisions happen locally. No external data transmission by default.
Fail-closed	Every missing configuration, invalid state, or absent policy defaults to the strictest safe posture.
Tamper-evident	SHA-256 hash-chained audit log — any modification to a single entry is detectable.
Content-minimal	Only file paths and metadata are persisted. No code content, diffs, or selection text is stored.
Explicit authority	Every decision packet carries a cryptographic authority assertion validated at construction time.

Data Flow


VS Code Save (Ctrl+S)
Save Hook Interceptor
← Extension intercepts before disk write
Rule Engine (Local Only)
← Pattern-match file path against risk rules
Decision Tier Classification
← ALLOW / WARN / REQUIRE_PLAN / BLOCK
Operator Prompt
← Human acknowledgment if required
Audit Log (.arc/)
← Append-only, hash-chained JSONL
Disk Write
← Save completes

2. Threat Model

2.1 Threats Addressed

T1: AI-Generated Code Without Oversight
Risk: An AI assistant generates a change to an authentication module. Without review, the change could introduce a vulnerability.

Mitigation: ARC XT classifies the save event against risk rules. Auth-related paths (`auth/`, `session/`) trigger a `REQUIRE_PLAN` decision, demanding a linked governance blueprint before the save proceeds.

Residual Risk: The blueprint system verifies documentation exists, not that the code itself is safe. ARC XT is not a code reviewer — it is an accountability layer.

T2: Audit Log Tampering
Risk: An attacker modifies audit entries to cover unauthorized changes.

Mitigation: Every audit entry is linked to the previous via SHA-256 hash chain (`prev_hash` → `hash`). Any modification to a single entry invalidates all subsequent entries. Verification is available via the CLI verifier.

Residual Risk: Wholesale deletion of the entire `.arc/` directory cannot be detected by hash chain validation alone. This is documented as a known limitation (see Section 11).

T3: Secret Exfiltration via Audit Log
Risk: Sensitive data (API keys, tokens, credentials) appears in audit entries.

Mitigation: Secret redaction is applied to any transient text (the "excerpt") before packet construction:

- `.env` key-value pairs with values ≥4 characters → redacted
- Bearer tokens (`Bearer <value>`) → redacted
- PEM private key blocks → fully redacted

Additionally, **no code content is ever persisted** to the audit log. Only file paths, timestamps, risk flags, and decision metadata are stored.

Residual Risk: File paths themselves may reveal project structure, codenames, or internal naming conventions. Mitigated by local-only storage with no external transmission.

T4: Unauthorized Cloud Data Transmission
Risk: Code excerpts or metadata are transmitted to external services without operator consent.

Mitigation: Cloud lane is **disabled by default**. The route policy defaults to `RULE_ONLY` with both lanes off. Even in `CLOUD_ASSISTED` mode, the `cloud_data_class` defaults to `LOCAL_ONLY` and must be explicitly changed. Auto-saves fail closed to `RULE_ONLY`.

Residual Risk: If an operator explicitly configures cloud lanes and sets data class to `CLOUD_ELIGIBLE`, excerpts (≤160 chars, redacted) may be transmitted to configured model endpoints. This requires deliberate configuration changes.

T5: Webview Injection Attacks

Risk: Malicious content injected into extension webviews via XSS or script injection.

Mitigation: Strict Content Security Policy with per-instance nonce:

- `default-src 'none'` — nothing allowed by default
- `script-src 'nonce-{nonce}'` — only scripts with valid CSP nonce
- `connect-src 'none'` — no XHR, fetch, or WebSocket connections
- `form-action 'none'` — no form submissions
- `object-src 'none'` — no plugins or embedded objects
- `frame-src 'none'` — no iframes

All event handlers use `addEventListener` (no inline `onclick`).

Residual Risk: The CSP allows `style-src 'self' 'unsafe-inline'` for VS Code theme variable compatibility. This is a deliberate trade-off — inline styles are needed for dynamic theming but represent a minor attack surface.

T6: Authority Spoofing
Risk: An attacker fabricates a context packet with a falsified authority tag to bypass enforcement.

Mitigation: The authority tag (`LINTEL_LOCAL_ENFORCEMENT`) is asserted by trusted local code and validated on every packet construction. Context packet validation requires:

- All required fields present
- `heuristic_only` must be `true`
- Authority tag must match the expected value
- Data class must match route policy
- Packet hash must match canonical serialization
- Validation failure throws an error — enforcement is fail-closed

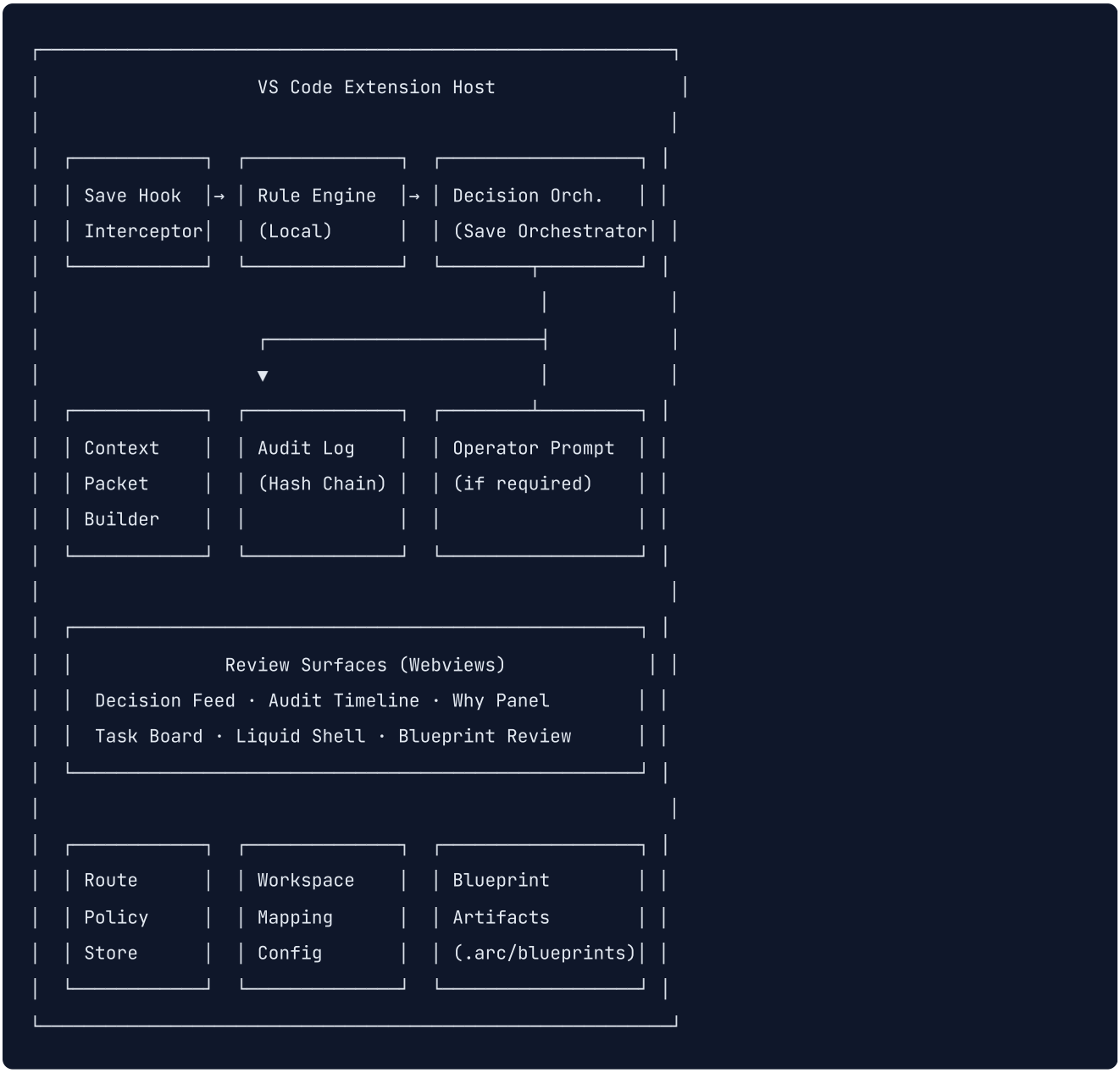
Residual Risk: The authority tag is a string constant, not a cryptographic signature. An attacker with access to the extension's runtime memory could theoretically forge packets. This is mitigated by the local-only execution model and VS Code's extension sandbox.

2.2 Threats NOT Addressed

Threat	Reason
Compromised VS Code installation	Outside extension scope
Malicious extension co-installed	Extension sandbox limitations
Supply chain attack on dependencies	Managed separately (see Section 3.3)
Physical access to filesystem	OS-level concern
Social engineering of operators	Governance process concern

3. Architecture Overview

3.1 Component Diagram



3.2 Data Storage

All ARC XT data is stored locally within the workspace at `.arc/`:

Path	Content	Format
`.arc/audit.jsonl`	Audit trail	JSON Lines, hash-chained
`.arc/audit.sqlite3`	Audit database	SQLite
`.arc/perf.jsonl`	Performance metrics	JSON Lines
`.arc/blueprints/`	Blueprint proofs	Markdown files
`.arc/router.json`	Route policy	JSON
`.arc/workspace-map.json`	Workspace mapping	JSON
`.arc/leases.jsonl`	Decision leases	JSON Lines
`.arc/overrides.jsonl`	Risk overrides	JSON Lines

3.3 Dependencies

Dependency	Version	Role	Audit Status
`ajv`	^8.18.0	JSON schema validation	Trusted — strict schema, no additional properties
`@types/vscode`	^1.90.0	Extension API types	Development only
`typescript`	^5.8.2	Compiler	Development only
`vitest`	^3.0.8	Test runner	Development only
`eslint`	^9.23.0	Linting	Development only

No runtime dependencies transmit data externally. All dependencies are pinned to specific versions in `package-lock.json`.

---

4. Audit Chain Integrity

4.1 Hash Chain Construction

Each audit entry is a JSON object with the following integrity fields:

```
{
  "event_type": "SAVE",
  "decision_id": "<SHA-256 of entry fields>",
  "ts": "2026-04-05T17:01:54.279Z",
  "file_path": "/absolute/path/to/file.ts",
  "decision": "ALLOW",
  "reason": "No Phase 1 heuristic rules were matched.",
  "risk_level": "LOW",
  "fingerprint": "<SHA-256 of file state>",
  "fingerprint_version": "lease.v1",
  "prev_hash": "<SHA-256 of previous entry>",
  "hash": "<SHA-256 of this entry including prev_hash>"
}
```

The `hash` field is computed as:

```
hash = SHA-256(serialized_entry_with_prev_hash)
```

Where `serialized_entry_with_prev_hash` is the entry's canonical JSON representation including the `prev_hash` of the preceding entry.

4.2 Chain Verification

To verify the integrity of an audit log:

```
cd your-workspace
npx arc-audit verify .arc/audit.jsonl
```

The verifier:

1. Reads each entry sequentially
2. Recomputes the `hash` from the entry fields
3. Compares with the stored `hash`
4. Checks that `prev_hash` matches the previous entry's `hash`
5. Reports any mismatches

4.3 Log Rotation

Audit logs are automatically rotated at a default threshold of 10 MB:

- `.arc/audit.jsonl` → active log
- `.arc/audit.jsonl.1` → most recent archived log
- `.arc/audit.jsonl.2` → older archived log

- ...and so on

Each rotated log maintains its own independent hash chain. Rotation does not invalidate existing chains.

---

5. Content Security Policy

5.1 Nonce Generation

A cryptographically random nonce is generated for each webview instance:

```
const nonce = crypto.randomBytes(16).toString('base64');
```

The nonce is embedded in the CSP header and in each `<script>` tag's `nonce` attribute.

5.2 CSP Directives

Directive	Value	Rationale
`default-src`	`'none'`	Deny everything by default
`script-src`	`'nonce-{nonce}'`	Only scripts with valid nonce
`style-src`	`'self' 'unsafe-inline'`	VS Code theme variables require inline styles
`img-src`	`'self' data:`	Extension assets and data URIs
`font-src`	`'self' data:`	Local fonts only
`connect-src`	`'none'`	**No network connections**
`frame-src`	`'none'`	No iframes
`object-src`	`'none'`	No plugins
`base-uri`	`'none'`	No `<base>` tag manipulation
`form-action`	`'none'`	No form submissions

5.3 CSP Enforcement

- Nonces are single-use — generated per webview instance
- All event handlers use `addEventListener` (no inline `onclick`)
- HTML content is sanitized via the `escapeHtml` function before injection
- No external fonts, scripts, or resources are loaded

---

6. Secret Redaction

6.1 Redaction Rules

Before any text is included in a context packet (for model evaluation), the following patterns are detected and redacted:

Pattern	Example	Action
`.env` variables	`DB_PASSWORD=secret123`	Value redacted: `DB_PASSWORD=[REDACTED]`
Bearer tokens	`Authorization: Bearer eyJhbG...`	Token redacted: `Authorization: Bearer [REDACTED]`
PEM private keys	`-----BEGIN RSA PRIVATE KEY----- -...`	Entire block redacted

6.2 Excerpt Limit

The excerpt — the portion of selected text used for model evaluation — is capped at **160 characters**. This limit serves two purposes:

1. Minimizes exposure of code context
2. Ensures the excerpt is sufficient for risk classification without leaking implementation details

6.3 No Persistence

Even the redacted excerpt is **never persisted** to the audit log. It is used only transiently during the classification phase and discarded.

---

7. Route Policy & Data Classification

7.1 Enforcement Modes

Mode	Description	Default
`RULE_ONLY`	Local heuristic rules only	✅ Yes
`LOCAL_PREFERRED`	Local model evaluation, falls back to rules	No
`CLOUD_ASSISTED`	Cloud model evaluation with local fallback	No

7.2 Lane Configuration

Lane	Default	Effect When Enabled
`local_lane_enabled`	`false`	Enables local model for evaluation
`cloud_lane_enabled`	`false`	Enables cloud model for evaluation

7.3 Data Classification

Data Class	Description	Default
`LOCAL_ONLY`	No data leaves the local machine	✅ Yes
`CLOUD_ELIGIBLE`	Data may be transmitted to configured endpoints	No
`RESTRICTED`	Stricter than LOCAL_ONLY — additional constraints	No

7.4 Governance Mode

Mode	Description
`ENFORCE`	Rules are actively enforced (blocks saves when required)
`OBSERVE`	Rules are logged but do not block saves

Default: **`ENFORCE`** — rules are actively enforced.

7.5 Config Validation

Route policy is validated against a strict JSON schema via Ajv:

- No additional properties allowed
- Required fields: `mode`, `local_lane_enabled`, `cloud_lane_enabled`, `cloud_data_class`
- Invalid config → falls back to `RULE_ONLY` with all lanes disabled

---

8. Context Packet Authority Model

8.1 Authority Tag

Every context packet carries an `authority_tag`:

```
{
  "authority_tag": "LINTEL_LOCAL_ENFORCEMENT",
  "heuristic_only": true,
  "data_class": "LOCAL_ONLY"
}
```

The authority tag asserts that:

- This packet was constructed by trusted local code
- The classification is heuristic-only (no full semantic understanding claimed)
- The data class matches the route policy

8.2 Packet Validation

Before a context packet is used for evaluation, it is validated:

Check	Condition	Failure Action
Required fields	All present	Throw error (fail-closed)
`heuristic_only`	Must be `true`	Throw error
`authority_tag`	Must match expected value	Throw error
`data_class`	Must match route policy	Throw error
Excerpt length	≤ 161 characters	Throw error
Hash	Must match canonical serialization	Throw error
`packet_id`	Must match hash prefix	Throw error

Every validation failure throws an error, and the save is blocked. This is the fail-closed design in action.

---

9. Fail-Closed Design

ARC XT is designed to fail safely. In every failure scenario, the system defaults to the most restrictive safe posture:

Failure Scenario	Default Behavior
Missing route policy	`RULE_ONLY`, all lanes disabled
Invalid route policy	`RULE_ONLY`, all lanes disabled
Missing workspace mapping	`LOCAL_ONLY`, no rules
Invalid context packet	Block save, throw error
Model unavailable	Fall back to heuristic rules
Config file unreadable	Fail to defaults
Audit log full	Rotate automatically
Blueprint not found	Block save (for REQUIRE_PLAN decisions)

This ensures that **ARC XT never silently weakens enforcement** due to configuration errors or missing files.

---

10. Privacy Guarantees

10.1 What Is Logged

Data	Logged?	Stored?	Transmitted?
File path	✅	✅	❌ (local only)
Timestamp	✅	✅	❌
Risk flags	✅	✅	❌
Matched rules	✅	✅	❌
Decision tier	✅	✅	❌
Directive ID	✅ (if linked)	✅	❌
Blueprint ID	✅ (if linked)	✅	❌
Route metadata	✅	✅	❌
Hash chain fields	✅	✅	❌

10.2 What Is NOT Logged

Data	Never Logged
File content	✅ Never stored
File diffs	✅ No before/after snapshots
Selection text	✅ Used transiently only, never persisted
User identity	✅ No username, email, or operator ID
Credentials/secrets	✅ Redacted before packet construction
External AI payloads	✅ No external AI request/response data

10.3 No Telemetry

ARC XT **does not collect or transmit telemetry**. There is:

- No usage analytics
- No crash reporting to external services
- No feature adoption tracking
- No performance monitoring to third parties

This is an explicit design decision — ARC XT is a local-first tool with no external dependencies.

10.4 Data Retention

Audit logs are retained indefinitely on the local filesystem. There is no automatic deletion. Manual deletion of audit entries breaks the hash chain and is detectable via verification.

---

11. Known Limitations

L1: Wholesale Audit Log Deletion

The hash chain detects modification of individual entries but **cannot detect deletion of the entire `.arc/` directory**. If an attacker removes all audit files, there is no cryptographic evidence of the deletion.

Mitigation: Organizations should implement filesystem-level monitoring on the `.arc/` directory. Future versions may integrate with external append-only stores.

L2: Absolute Path Disclosure

Audit entries contain absolute file paths, which may reveal:

- Project directory structure

- Internal codenames
- Developer workspace layout

Mitigation: All data is stored locally with no external transmission. Operators with high-security requirements should use workspace paths in isolated environments.

L3: Authority Tag Is Not Cryptographically Signed

The `LINTEL_LOCAL_ENFORCEMENT` authority tag is a string constant asserted by local code, not a cryptographic signature. An attacker with access to the extension's runtime memory could theoretically forge packets.

Mitigation: The VS Code extension sandbox limits this attack surface. Future versions may introduce cryptographic signing for context packets.

L4: No Cryptographic Binding to File Content

The audit log records that a save occurred for a specific file path, but does not cryptographically bind the decision to the actual file content. An operator could save different content than what was classified.

Mitigation: The fingerprint field captures file state at save time. Future versions may strengthen this binding.

L5: Extension Cannot Audit Other Extensions

ARC XT audits file saves within VS Code but cannot audit actions performed by other extensions that bypass the standard save lifecycle.

Mitigation: ARC XT hooks the standard VS Code save lifecycle. Extensions that write files directly to disk without triggering save events would bypass ARC XT. This is a known boundary condition.

---

12. Verification Procedures

12.1 Audit Log Verification

```
cd your-workspace
npx arc-audit verify .arc/audit.jsonl
```

Expected output: all entries verified, no hash mismatches.

12.2 Route Policy Verification

```
cat .arc/router.json
```

Verify:

- `mode` is a recognized value (`RULE_ONLY` , `LOCAL_PREFERRED` , or `CLOUD_ASSISTED`)
- `cloud_data_class` is `LOCAL_ONLY` unless explicitly changed
- Both lanes are disabled unless intentionally enabled

12.3 CSP Verification

1. Open any ARC XT webview panel
2. Open Developer Tools (`Help → Toggle Developer Tools`)
3. Check the Console for CSP violations — there should be none
4. Check Network tab — no outbound connections should be present

12.4 Package Integrity

```
# Verify the VSIX contains only expected files
unzip -l arc-audit-ready-core-0.1.13.vsix | grep -v 'dist/\|package\.json\|README\|LICENSE\|rules/\|
```

Expected: no output (no unexpected files).

12.5 Build Reproducibility

```
git clone https://github.com/habrahgithub/arc-extension.git
cd arc-extension
npm install
npm run build
npm run pack
```

Compare the generated `.vsix` hash with the distributed release.

---

13. Release History

Version	Date	Security Notes
0.1.13	Apr 2026	Clean baseline — full ARC XT identity, packaging hygiene, Liquid Shell preview
0.1.12	Apr 2026	Task Board activity bar, decision visibility layer
0.1.11	Apr 2026	Phase 7 completion — precision hardening, false positive scoring
0.1.10	Apr 2026	Phase 6.8 — end-to-end evidence pack

For detailed security audit history, see the `.arc/audit.jsonl` file in any governed workspace.

This white paper corresponds to ARC XT v0.1.13. It documents the security architecture as implemented, not aspirational capabilities. Every claim herein is verifiable against the source code.

For security inquiries, contact the maintainers via GitHub Issues.