EN

Practical Guide

OpenClaw: Secrets in AI Context

Credentials, API Keys and Vault

Vault, dynamic injection and secret protection for AI agents

Mars 2026

OpenClaw Series — Post 4/7

Never Expose Your Secrets in AI Context

Credentials, API Keys and Vault — Complete BOTUM Architecture

1. The Fundamental Paradox

For an OpenClaw agent to be truly autonomous, it needs access to the services it acts on: GitHub API key, SMTP token, Vaultwarden password, SSH access. The paradox: the agent needs these secrets to operate — but how you provide them is critical.

Passing an API key directly in a message, hardcoding it in a workspace script, or storing it in MEMORY.md amounts to exposing it in the AI context — an attack surface in its own right.

2. Why the AI Context Is an Attack Surface

History Logs

OpenClaw keeps conversation history. Telegram keeps message history. If an API key passes through a message or a versioned workspace file, it persists in places no one thinks about. A git log becomes an exfiltration vector.

Model Responses

LLMs can reproduce data they've seen in their context. Several documented incidents involve models "citing" security tokens that were sitting in their input context.

Prompt Injection

If an agent processes external content (emails, web pages, GitHub issues), a malicious actor can inject instructions to exfiltrate secrets present in the context. Documented by OWASP LLM Security Top 10.

Memory Persistence

MEMORY.md, CODEX.md, state files — everything that persists in the workspace can be re-read by the agent in future sessions, or by other agents in the network.

3. Classic Mistakes

The API Key in MEMORY.md

The agent notes the SMTP key in MEMORY.md to avoid asking every session. Result: the key is in the permanent context of all sessions and in Git history.

The Password in a Telegram Message

Password sent via Telegram to "move fast." Stored on Telegram's servers, in the OpenClaw session history, and potentially logged locally — at least 3 uncontrolled locations.

Hardcoded in a Workspace Script

API_KEY = "sk-..." hardcoded in a Python script in a Git workspace. A git log -p will find the key even after deletion.

Credentials in Config Files

openclaw.json or config.yaml with tokens in plain text. An accidental commit or file share = guaranteed leak.

4. The Solution: Vault + Dynamic Injection

Golden rule: secrets only live in the vault and are retrieved only at execution time. Never hardcoded, never in context, never in a versioned file.

Recommended Pattern (shell)

```
# Good pattern: retrieval at execution time TOKEN=$(bw get password "api-service-x") curl
-H "Authorization: Bearer $TOKEN" https://api.service-x.com/ unset TOKEN # immediate
cleanup
```

Pattern to Absolutely Avoid

```
# BAD PATTERN — Never do this TOKEN="sk-abc123xyz" # hardcode = permanent risk curl -H
"Authorization: Bearer $TOKEN" https://api.service-x.com/
```

With the vault as the single source of truth, rotating a secret is trivial: change the value in Vaultwarden, and all scripts automatically pick up the new value.

5. Environment Variables vs Vault — When to Use Which

Use Case	Env Variables	Vault
Infrastructure secrets (Docker, systemd)	✓ Suitable	Possible
Secrets shared across multiple agents	✗ Risky	✓ Recommended
Frequently rotated credentials	✗ Cumbersome	✓ Ideal
Secrets injected into AI context	✗ Avoid	✗ Avoid
Ephemeral tokens (OAuth, JWT)	✓ Suitable	Possible

Final rule: no secret should ever be injected into the AI agent's context — neither via environment variable nor via vault. The agent invokes a tool that retrieves and uses the secret in isolation, without that secret transiting through the LLM context window.

6. OpenClaw Best Practices

The Git Workspace: Code + Config, Never Secrets

The OpenClaw workspace is versioned by Git. Everything committed becomes permanent (even after deletion, git log retains a trace).

What can be committed: code, scripts (without credentials), config files without sensitive values, templates. What must never be committed: API keys, passwords, OAuth tokens, private certificates, private SSH keys.

Strict .gitignore — Minimum List

```
# Secrets — NEVER in Git .env | .env.* | *.key | *.pem | *.p12 secrets.json |
config.local.yaml | credentials.json # OpenClaw specific MEMORY.md | openclaw.json |
*.session
```

Regular Secret Rotation

According to CyberArk's 2025 study, 68% of security incidents involving credentials relate to secrets that were never rotated. BOTUM recommendation: 90 days for infrastructure tokens, 30 days for high-value API access keys.

7. Real Scenario: The BOTUM Architecture

Layer 1: Self-Hosted Vaultwarden

A Vaultwarden instance runs on us-srv-dck-01, accessible only from the internal network. Collections: infrastructure, external-apis, openclaw-agents, smtp.

Layer 2: Isolated Injection Script

Each agent has a vault-inject.sh helper script that connects to the vault via bw CLI with an ephemeral token, retrieves the specific secret, injects it as a temporary variable, executes the action, and immediately cleans up.

Layer 3: No Persistence in Context

Agents have access to the names of secrets in the vault ("github-token-fdbot"), not the secrets themselves. The secret never transits through the LLM context window.

Layer 4: Audit and Alerts

KNOX, the security agent, generates a weekly report on credential usage: access by agent, anomaly detection, rotation reminders.

8. Checklist — 10 Rules to Never Leak a Secret

1. **1.** Every secret lives in the vault — Vaultwarden, HashiCorp Vault, or equivalent. Nowhere else.
2. **2.** Zero secrets in MEMORY.md, CODEX.md, or any workspace file — persistent and read by all agents.
3. **3.** Zero secrets in Telegram/Slack/email messages — messaging apps are not secret managers.
4. **4.** Zero hardcoded secrets in scripts — even in local dev, even "temporarily."
5. **5.** Exhaustive .gitignore — include all sensitive file patterns before the first commit.
6. **6.** Dynamic injection only — the secret is retrieved and used in memory, never written.
7. **7.** Principle of least privilege — each agent only accesses the secrets it strictly needs.
8. **8.** Rotation on schedule — 90 days for infra, 30 days for high-value APIs.
9. **9.** Regular access audits — who accessed what, when, from which agent.
10. **10.** Clean up revoked secrets — an expired or revoked token is immediately deleted from the vault.

Conclusion

Managing secrets in an AI agent network is one of the most silent risk vectors in modern automation. BOTUM's approach — central vault, dynamic injection, no secrets in AI context, rotation on schedule — eliminates an entire category of risks permanently.

→ Post 5: Configuring your first specialized agents — JARVIS, HERMÈS, CHRONOS