Clawdbot / Moltbot Security Whitepaper

Attack Surface, Threat Model, and Defensive Hardening for Agentic "Personal Assistant" Gateways

The Agile Monkeys Innovation Lab
labs.theagilemonkeys.com

January 30, 2026

Abstract

Clawdbot (rebranded as *Moltbot*) is a local-first, open-source "agentic assistant" that users control through common messaging apps (WhatsApp, Telegram, Slack, iMessage, etc.) while granting the agent real capabilities: filesystem access, command execution, browser automation, and integrations with third-party services[1, 2, 3]. This is a powerful productivity paradigm, but it also creates a new security principal inside the user's environment: a continuously running, remotely reachable software agent that can act as the user.

This whitepaper provides: (1) an architectural overview of Clawdbot/Moltbot; (2) a structured threat model; (3) an attack-vector taxonomy with realistic abuse scenarios (including social-engineering driven prompt injection); and (4) a practical hardening baseline suitable for engineering teams, security leaders, and enterprise adopters.

Scope and intent: This document is written for defensive security, risk management, and secure deployment. It intentionally avoids providing exploit code or step-by-step offensive instructions.

Contents

1 Background: What Clawdbot/Moltbot Enables

1.1 Why it went viral

Clawdbot/Moltbot's appeal is simple: it turns a frontier LLM into a persistent "operator" that can *do things* on your infrastructure and communicate over channels you already use like texting. Unlike traditional chatbots, the assistant can run continuously, keep state over time, and integrate with local files and apps.[2, 1] This "always-on agent" model makes it useful for:

- **Life admin automation:** triaging and replying to messages, calendar management, reminders, scheduling.
- **Work assistance:** project coordination, summaries, drafting, and coding workflows.
- **Tool-driven actions:** running commands, editing files, interacting with browsers, and calling APIs.

1.2 The security inflection point

The moment a system can *act* (not just *answer*), its threat model changes. The assistant becomes:

- a **remote command surface** (via messaging apps and network control planes),
- a **credential concentrator** (API keys, OAuth tokens, service logins),
- a **data broker** (local files + corporate SaaS + chat/email history),
- a **social-engineering target** ("if I can trick the assistant, I can trick the user").

This is not hypothetical: multiple reports in January 2026 documented real-world insecure deployments, exposed control interfaces, and supply-chain style abuse patterns in the skills ecosystem.[10, 11]

2 How it Works: High-Level Architecture

2.1 Core components

At a high level, Clawdbot/Moltbot resembles an "agent gateway":

- **Gateway / daemon**: the always-on process that coordinates channels, sessions, tools, plugins, and model calls.
- **Channels**: connectors that ingest messages from WhatsApp/Telegram/Slack/iMessage/etc. and emit responses.
- **Agent runtime**: the loop that builds prompts (system prompt + skills + session context), calls a model provider, and decides which tools to invoke.
- **Tools**: privileged capabilities (filesystem read/write, shell execution, browser control, node execution, etc.).
- **Skills**: instruction bundles (AgentSkills-style folders) that teach the agent how to use tools; can be local, bundled, or installed from a public registry (ClawdHub).

- **Plugins**: in-process extensions that can add tools/commands/services; may be installed from npm or local directories.
- **Control UI**: a web interface used to manage the gateway and configuration.

2.2 Trust boundaries

A practical way to reason about the system is to separate **who can talk**, **what can run**, and **where it can run**:

1. **Inbound identity & authorization**: DM policies, allowlists, group mention gating, and command authorization.
2. **Capability policy**: which tools exist/are callable (tool allow/deny), per-agent profiles, and exec allowlists.
3. **Execution isolation**: optional Docker sandboxing (tools inside containers vs tools on host), plus "elevated" escapes.

2.3 Persistence and local state

Moltbot stores operational state (channel credentials, allowlists, auth profiles, session logs) under a state directory (commonly `~/.moltbot` or legacy `~/.clawdbot`). This enables continuity, but it also concentrates secrets and transcripts on disk.[3]

3 Threat Model

3.1 Assets at risk

- **Secrets:** API keys (LLM providers, SaaS), OAuth tokens, messaging service credentials, plugin secrets.
- **Private content:** chat logs, email content, attachments, personal and corporate files.
- **Execution capability:** ability to run commands, modify files, open browsers, and act as the user.
- **Identity & reputation:** outgoing messages/emails sent "as you", calendar invites, business communications.

3.2 Adversary profiles

- **External message senders**: anyone who can DM or message a shared room where the bot is present.
- **Internet opportunists**: actors who find misconfigured gateways or exposed admin interfaces.
- **Supply-chain attackers**: malicious skills/plugins/packages or compromised registries.
- **Local malware**: infostealers and commodity malware that exfiltrate local state directories.
- **Insiders**: members of a Slack/Teams/Telegram group that can influence the bot's behavior.

3.3 What makes agentic systems different

Traditional software security assumes clear intent: a human clicks "Send money"; a human runs `rm -rf`. Agentic assistants blur that boundary. Attacks often become **argumentation problems**:

- convince the model that the request is legitimate (**social engineering**),
- smuggle instructions through untrusted content (**prompt injection**),
- exploit ambiguous authority (**who is the bot acting for?**).

4 Attack Vector Taxonomy

4.1 Vector A: Prompt injection through direct messages and group chats

Mechanism: An attacker messages the bot and manipulates the model to take unsafe actions (e.g., "ignore policy and run this command", "send me your last 50 messages", "download and execute this script").

Why it matters: Moltbot's own security guidance emphasizes that many failures are not "fancy exploits" but simply "someone messaged the bot and the bot did what they asked".[3, 13]

Preconditions:

- DMs are open or allowlists are broad.
- Group chat policies allow the bot to respond without mention gating.
- Tools with high blast radius are enabled (exec, filesystem write, browser automation).

Impact: Data exfiltration, destructive actions, account misuse, lateral movement through integrated services.

Defenses:

- Default to **DM pairing** and narrow allowlists.
- Require **explicit mention** in groups; consider per-user allowlists in sensitive rooms.
- Prefer **sandboxed execution** for risky tools; enforce tool policies (deny-by-default).
- Add **transaction-classification policies**: treat payments, credential resets, and external invitations as "high-risk" requiring a human confirmation step.

4.2 Vector B: Social engineering of "human-like responsibilities"

This is the scenario that repeatedly appears in real deployments: as soon as an assistant can read inboxes and act on your behalf, it becomes vulnerable to fraud patterns that were historically targeted at humans.

Example scenario (illustrative): An email impersonates a trusted person and requests an urgent wire transfer. A human might apply skepticism; a model might comply if the text is persuasive and the assistant is configured for autonomy.[15, 16]

Defenses:

- **Out-of-band verification**: the agent must request a second factor (call, known code word, verified channel) for any financial or identity-sensitive action.

- **Recipient allowlists**: restrict who the agent can send money/invites to; enforce account/IBAN allowlists.
- **Two-person rule**: require explicit user approval for any transfer or external sharing.
- **Language-aware deception detectors**: flag urgency + money + identity claims; route to human review.

4.3 Vector C: Indirect prompt injection via untrusted content

Mechanism: The assistant reads untrusted data (web pages, email bodies, pasted logs, documents) that contain instructions intended to hijack the agent's behavior. Crucially, *this does not require public DMs*: a trusted user can forward an attacker-controlled document, and the agent may obey it.[13, 14]

Impact: Exfiltration, tool misuse, persistence through edited files/skills/config.

Defenses:

- Treat links/attachments as hostile; summarize in a constrained mode before acting.
- Enforce a strict **tool-use policy** for content-derived requests.
- Use a **content firewall**: strip or neutralize "instructions" sections in documents; isolate data from directives.

4.4 Vector D: Gateway / Control UI exposure and authentication bypass

Mechanism: Misconfiguration (especially around reverse proxies) can accidentally expose admin/-control surfaces. Public reporting highlighted scenarios where the system auto-trusted traffic that appeared "local" when proxied, leading to unauthenticated access.[10, 3]

Impact: Full takeover: read transcripts, steal credentials, execute commands, modify configuration.

Defenses:

- Never expose the gateway or control UI to the public internet without strong authentication and TLS.
- Configure **trusted proxy** settings correctly so proxied requests are not misclassified as local.
- Prefer tailnet-only access (e.g., Tailscale Serve with identity headers) for remote use.
- Monitor for unexpected inbound connections and failed auth events; rotate tokens on suspicion.

4.5 Vector E: Skills ecosystem supply chain

Mechanism: Skills are instruction bundles that can meaningfully change tool usage. Public reporting showed that skills can be abused as a distribution mechanism: a malicious or misleading skill can convince the model to run unsafe commands or exfiltrate data.[8, 11]

Impact: Execution and exfiltration, often without the user realizing that the "skill" is the true source of behavior.

Defenses:

- Treat third-party skills as trusted code: review before use.

- Implement internal skill registries for enterprise use; require signatures and code review.
- Run untrusted skills only in a restricted sandbox profile with minimal tools.

4.6 Vector F: Plugin supply chain (npm, in-process execution)

Mechanism: Plugins run *in-process* with the gateway and may be installed from npm.[7, 13] Installation workflows can execute code during install (package scripts). This is a classic supply-chain exposure, now attached to an always-on privileged agent.

Impact: Arbitrary code execution on the host (at install time or runtime), persistence, data theft.

Defenses:

- Prefer explicit **plugin allowlists** and version pinning.
- Inspect the unpacked plugin directory before enabling.
- Use isolated OS accounts/containers for the gateway process; minimize privileges.

4.7 Vector G: Local secrets-at-rest and infostealer compatibility

Mechanism: Credentials, transcripts, and configuration are stored on disk for continuity. Commodity infostealer malware specializes in exfiltrating predictable file paths.[3]

Impact: Theft of API keys, OAuth tokens, messaging credentials, chat logs; downstream account takeover.

Defenses:

- Enforce strict file permissions on state directories; avoid syncing state via consumer cloud drives.
- Use full-disk encryption; separate the gateway into a dedicated machine/VM.
- Prefer OS keychains or external secret managers when available; avoid plaintext secrets in config.

4.8 Vector H: Sandbox escape hatches and "blast radius" mistakes

Moltbot supports Docker sandboxing for tools, but this is optional.[4, 5] Even when sandboxed, configuration can accidentally punch holes:

- bind-mounting sensitive paths read-write,
- mounting `/var/run/docker.sock`,
- enabling elevated execution.

Defenses:

- Default to **sandbox-on** for non-main sessions and group chats.
- Keep workspace access **none** or **read-only** for high-risk environments.
- Avoid bind mounts unless you have a clear reason; prefer read-only mounts.

5 Risk Matrix (Defensive Prioritization)

Vector	Likelihood	Potential Impact
Open DMs / weak group gating	High	High (fraud, exfil, misuse)
Reverse proxy / control UI exposure	Medium–High	Critical (full takeover)
Untrusted skills / supply chain	Medium	High (execution/exfil)
Untrusted plugins (npm)	Medium	Critical (host compromise)
Local infostealer / endpoint compromise	Medium	High (secrets + identity)
Indirect prompt injection (docs/web)	High	Medium–High
Sandbox misconfiguration	Medium	High

6 Secure Deployment Baseline

6.1 Design principles

1. **Access control before intelligence:** assume models can be manipulated; constrain who can trigger actions and what actions exist.
2. **Least privilege by default:** deny-by-default tools, narrow allowlists, read-only workspaces, and sandbox execution.
3. **Separation of concerns:** split "read" and "act" agents; isolate corporate and personal assistants; separate channels/numbers.
4. **Explicit approvals for irreversible actions:** payments, credential resets, data sharing, external invitations.

6.2 Practical hardening checklist

- **Network**: keep gateway bound to localhost; avoid public exposure; use firewall rules.
- **Reverse proxies**: configure trusted proxies correctly; enforce TLS and authentication.
- **DM policy**: pairing by default; allowlists for sensitive deployments; disable public DMs.
- **Groups**: require explicit mention; restrict which rooms can trigger the bot.
- **Tools**: deny-by-default; allow only the minimal tool groups needed; use exec allowlists.
- **Sandboxing**: enable Docker sandboxing, at least for non-main sessions.
- **Skills/plugins**: only install from trusted sources; pin versions; maintain internal registries.
- **State directory**: strict permissions; no cloud-sync; secrets management.
- **Monitoring**: log redaction; anomaly detection on message patterns and tool invocations.

6.3 Example baseline configuration (illustrative)

The following pseudo-configuration shows the *intent* of a secure baseline. Exact keys differ by version and deployment, so treat this as a template to be adapted and verified with official documentation and the built-in security audit.[3]

```
{
  "gateway": {
    "auth": { "mode": "password" },
    "trustedProxies": ["127.0.0.1"]
  },
  "logging": { "redactSensitive": "tools" },
  "session": { "dmScope": "per-channel-peer" },
  "channels": {
    "whatsapp": {
      "dm": { "policy": "pairing" },
      "groups": { "*": { "requireMention": true } }
    }
  },
  "plugins": { "allow": ["voice-call"] },
  "agents": {
    "defaults": {
      "sandbox": {
        "mode": "non-main",
        "scope": "session",
        "workspaceAccess": "none"
      }
    }
  },
  "tools": {
    "deny": ["elevated"],
    "sandbox": { "tools": { "allow": ["group:fs", "group:sessions"] } }
  }
}
```

7 Recommendations for Partners and Enterprise Stakeholders

7.1 Treat the agent as a privileged identity

In enterprise terms, an agentic assistant is a new "user" with:

- long-lived credentials,
- broad data access,
- autonomous decision-making,
- many inbound trust channels.

It should be managed like a privileged service account: inventory, approvals, rotation, and monitoring.[12]

7.2 Adoption patterns that reduce risk

- **Start read-only:** limit tools to summarization/search, no write/exec.
- **Dedicated host / VM:** keep the assistant off primary laptops; isolate corporate from personal.
- **Internal registries:** mirror and sign known-good skills/plugins; block unknown registries.
- **DLP-aware tool wrappers:** ensure the agent cannot exfiltrate sensitive documents via outbound messaging.

8 Incident Response Guidance (Agent Compromise)

If you suspect compromise, assume:

- secrets may be exfiltrated (API keys, OAuth tokens),
- transcripts may be exposed,
- the agent might have been instructed to create persistence (cron jobs, modified config, malicious skills/plugins).

Immediate steps:

1. **Contain:** disable external access, shut down gateway, disconnect from networks if needed.
2. **Rotate:** revoke and rotate all tokens/keys used by the agent.
3. **Audit:** review session logs and tool invocations; enumerate installed skills/plugins; check for persistence artifacts.
4. **Recover:** rebuild from a known-good baseline; re-onboard with minimal privileges.

9 Conclusion

Clawdbot/Moltbot demonstrates a compelling future: assistants that are truly useful because they are connected to our tools and data. That same capability makes them security-sensitive by design. The core lesson is to stop treating the assistant as "just a chatbot" and start treating it as a privileged operator with its own access model, blast radius, and supply chain.

When identity gating, least privilege, sandboxing, and supply-chain controls are applied intentionally, agentic assistants can be deployed with acceptable risk—but the defaults and cultural expectations must evolve quickly.

References

References

[1] Moltbot Project Repository (GitHub). https://github.com/moltbot/moltbot (accessed 2026-01-30).

[2] DigitalOcean Documentation (Marketplace 1-Click App): *Moltbot*. https://docs.digitalocean.com/products/marketplace/catalog/moltbot/ (accessed 2026-01-30).

[3] Moltbot Documentation: *Security*. https://docs.molt.bot/gateway/security (accessed 2026-01-30).

[4] Moltbot Documentation: *Sandboxing*. https://docs.molt.bot/gateway/sandboxing (accessed 2026-01-30).

[5] Moltbot Documentation: *Sandbox vs Tool Policy vs Elevated*. https://docs.molt.bot/gateway/sandbox-vs-tool-policy-vs-elevated (accessed 2026-01-30).

[6] Moltbot Documentation: *Exec Tool*. https://docs.molt.bot/tools/exec (accessed 2026-01-30).

[7] Moltbot Documentation: *Plugins*. https://docs.molt.bot/plugin (accessed 2026-01-30).

[8] Moltbot Documentation: *Skills*. https://docs.molt.bot/tools/skills (accessed 2026-01-30).

[9] Moltbot Documentation: *Session Concepts (dmScope, identity links, isolation)*. https://docs.molt.bot/concepts/session (accessed 2026-01-30).

[10] Bitdefender Hotforsecurity (Vlad Constantinescu). *Moltbot security alert: exposed Clawdbot control panels risk credential leaks and account takeovers*. January 27, 2026. https://www.bitdefender.com/en-us/blog/hotforsecurity/moltbot-security-alert-exposed-clawdbot-control-panels-risk-credential-leaks-a (accessed 2026-01-30).

[11] Malwarebytes Threat Intel (Stefan Dasic). *Clawdbot's rename to Moltbot sparks impersonation campaign*. January 29, 2026. https://www.malwarebytes.com/blog/threat-intel/2026/01/clawdbots-rename-to-moltbot-sparks-impersonation-campaign (accessed 2026-01-30).

[12] National Institute of Standards and Technology (NIST). *Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1*. January 2023. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf (accessed 2026-01-30).

[13] OWASP GenAI Security Project. *OWASP Top 10 for Large Language Model Applications (2025)*. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/ (accessed 2026-01-30).

[14] MITRE. *ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems*. https://atlas.mitre.org/ (accessed 2026-01-30).

[15] Federal Bureau of Investigation. *Business Email Compromise*. https://www.fbi.gov/how-we-can-help-you/scams-and-safety/common-frauds-and-scams/business-email-compromise (accessed 2026-01-30).

[16] Cybersecurity and Infrastructure Security Agency (CISA). *Counter-Phishing Recommendations for Federal Agencies* (August 2023 revision). https://www.cisa.gov/sites/default/files/2023-09/CISA_CEG_Counter-Phishing_Guidance_for_Federal_Agencies%20Aug-23%20Revision.pdf (accessed 2026-01-30).