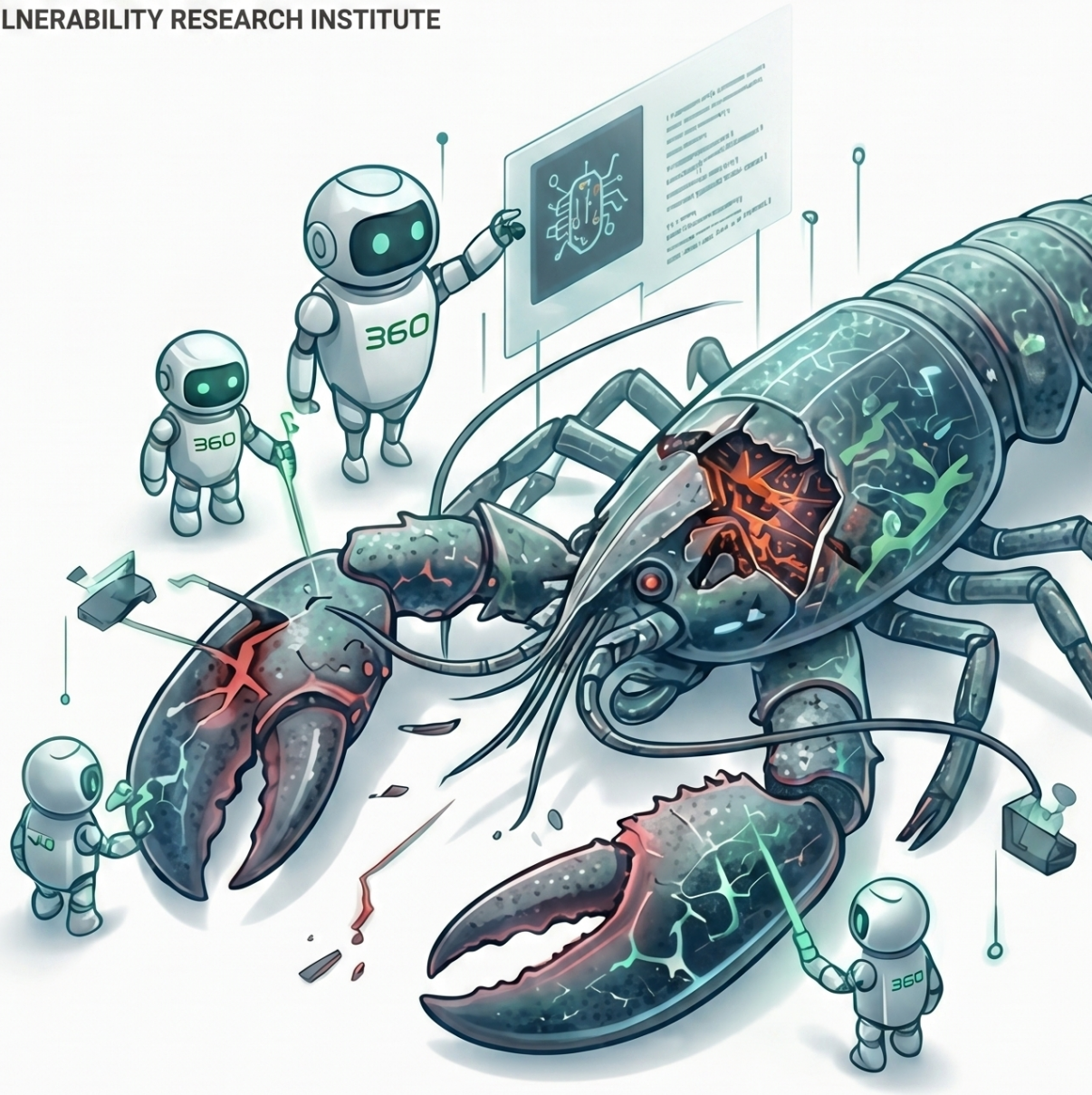
360漏洞研究院
VULNERABILITY RESEARCH INSTITUTE



360漏洞挖掘智能体实践

—— OpenClaw生态安全风险分析

二〇二六年五月

目 录

一、 概述

OpenClaw 核心能力的不断成熟与全面开放，引发了业界的关注热潮。得益于其底层架构的灵活性与高扩展性，OpenClaw 不仅支撑了大量基于它的二次开发，更启发了众多团队借鉴其理念开启独立自研，推动生态向多极化方向发展。

这种繁荣同时也伴随着产品形态的能力转变。当前的 Claw 系应用正逐渐打破传统工具“被动响应”的局限，向着具备自主决策能力的 Agent 跨越。这种能力跃迁不可避免地要求系统赋予其更复杂的接口与更宽泛的权限边界。当这些高权限、高自治的实体运行在不可信的网络环境中，或是面临潜在的恶意攻击时，失控的风险将被急剧放大。面对此类具备高度主观能动性的新型应用目标，传统的安全测试手段往往难以有效覆盖其动态逻辑与模糊边界。

基于此，本报告开展了一次深度的“漏洞挖掘智能体实践”。360 漏洞研究院引入自研的漏洞挖掘智能体，以“Agent 对抗 Agent”的创新范式，对 OpenClaw 生态产品展开了系统性的安全分析。借助漏洞挖掘智能体高效的全流程自动化漏洞挖掘能力，我们成功突破了复杂应用的测试瓶颈，累计挖掘出安全漏洞 20 余个，涵盖远程代码执行、认证绕过、权限提升、信息泄露等多种高危漏洞类型。

图 1-1 OpenClaw 生态图概览

在对智能体在实战中捕获的丰富漏洞数据与攻击链路进行归纳分析后，我们

发现，由于 OpenClaw 生态演进的独特性，其潜在的安全风险呈现出多样化、普遍性与可传播性，并深度渗透至各类衍生产品之中。本报告将结合此次智能体实践的成果，将 OpenClaw 生态划分为三大核心分析维度：OpenClaw 架构特征与防护失效、二次开发中的安全债传递、以及开源自研的安全挑战。围绕上述维度，本报告将系统性地剖析不同应用场景下潜藏的安全风险，旨在为 Claw 生态的持续、健康与积极发展夯实安全底座，贡献实质力量。

二、 漏洞列表

本研究共覆盖 OpenClaw 核心及 10 款 Claw 衍生产品，经系统性分析共确认 23 处独立安全漏洞。所有漏洞均已反馈至相关厂商与开发者跟进修复，并上报国家信息安全漏洞库（CNNVD）、国家信息安全漏洞共享平台（CNVD）等权威机构。下表按产品归属和漏洞类型进行汇总：

序号	产品名称	漏洞类型	CNNVD	危害程度
1	OpenClaw	信息泄露	CNNVD-2026-50773322	高危
2	OpenClaw	信息泄露	CNNVD-2026-45978429	中危
3	OpenClaw	拒绝服务	CNNVD-2026-18369523	高危
4	OpenClaw	授权绕过	CNNVD-2026-24759492	中危
5	LobsterAI	远程任意文件读取	CNNVD-2026-71027986	中危
6	LobsterAI	路径穿越	CNNVD-2026-82655690	高危
7	AutoClaw	远程命令执行	CNNVD-2026-01108369	高危
8	ClawX	信息泄露	CNNVD-2026-31313202	中危
9	CoPaw	远程命令执行	CNNVD-2026-22285599	严重
10	CoPaw	远程命令执行	CNNVD-2026-43521646	严重
11	Nanobot	提示词注入	CNNVD-2026-78883522	中危
12	Nanobot	服务端请求伪造	CNNVD-2026-20334794	中危
13	Nanobot	访问控制绕过	CNNVD-2026-20084544	中危
14	Nanobot	访问控制绕过	CNNVD-2026-96922764	中危
15	Nanobot	服务端请求伪造	CNNVD-2026-22577557	中危
16	Nanobot	访问控制绕过	CNNVD-2026-68396869	高危
17	Nanobot	路径穿越	CNNVD-2026-86246607	高危

序号	产品名称	漏洞类型	CNNVD	危害程度
18	Molili	远程命令执行	CNNVD-2026-55617819	高危
19	PicoClaw	访问控制绕过	CNNVD-2026-99907553	高危
20	PicoClaw	访问控制绕过	CNNVD-2026-83686554	高危
21	QClaw	逻辑缺陷	CNNVD-2026-78347890	中危
22	Winclaw	认证绕过	CNNVD-2026-30781760	中危
23	ZeroClaw	认证绕过	CNNVD-2026-86380766	高危

表格所示漏洞涵盖远程代码执行、认证与访问控制、越权文件操作、提示词注入与信息泄露、拒绝服务类等漏洞类型。这一分布特征与 OpenClaw 及其衍生产品的形态相符：多入口、高权限、重本地服务、弱组件间隔离。

为避免安全风险外溢及防止潜在的恶意利用，本报告中涉及的所有具体漏洞案例均进行匿名化处理。

三、 OpenClaw 架构特征与防御失效分析

在 OpenClaw 开源仓库中，与其 37 万 GitHub Stars 同样引人注目的，是其居高不下的安全公告数量。截至 2026 年 5 月 11 日，OpenClaw GitHub 已累计披露超过 535 个安全公告，在同期开源项目中处于显著高位。这客观反映出一个事实：在 OpenClaw 功能日益丰满的同时，其暴露的攻击面正被快速拓宽，深层的安全隐患也在同步积聚。

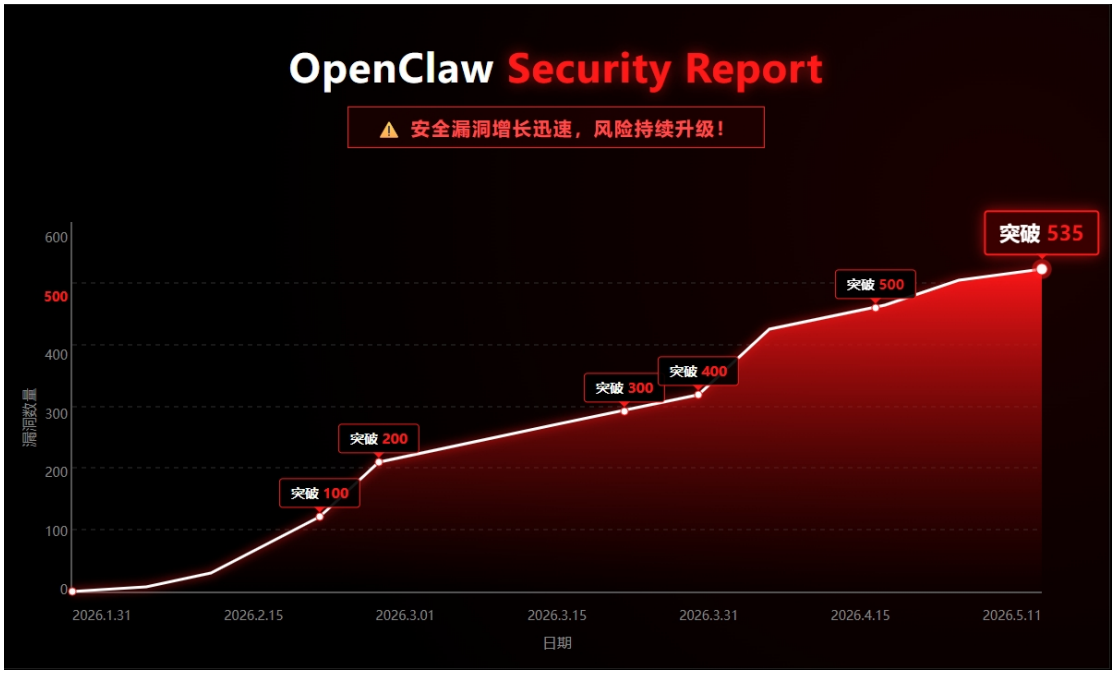


图 3-1 OpenClaw 生态安全报告增长曲线图

从时间维度观察，这些安全风险的暴露正呈现出明显的加速趋势。在项目早期，安全公告大多以“周”为单位零星发布；然而进入 2026 年第一季度后，其披露频率已骤然攀升至日均 4 个以上。这种由疏到密的转变，其深层原因在于当前生态中普遍存在“能力扩展优先于安全约束”的开发倾向，激进的功能迭代与相对滞后的安全内建，最终催生了日益严峻的系统性风险。

3.1 架构特征

为了更好理解这些安全公告所反映的漏洞根因与攻击场景，需结合 OpenClaw 的核心运行架构进行分析。作为一个以本地资源操控为设计哲学的 AI Agent 网关，OpenClaw 的运作依赖于三个相互嵌套的基础组件：

本地工具层（Tool Calling Layer）：负责将大模型的决策转化为对本地资源的实际操作，包括文件读写、命令行执行、浏览器自动化、代码动态编译等。该层直接触及操作系统核心，必须严格验证操作主体身份并限定操作权限范围。

网络连接层（Network Layer）：支撑 Agent 与外部世界的双向数据交换。一方面，Agent 需要主动抓取网页、调用外部 API、下载资源；另一方面，系统又在本地暴露 HTTP 服务、WebSocket 连接、RPC 端口以接收外部请求。该层必须确保入口合法与出口可控性。

决策核心层（Control Plane）：以大型语言模型为中枢，将自然语言指令转化为任务计划并调度工具执行。该层需要持续接收并理解多源输入（用户指令、网页内容、工具反馈），必须确保指令来源可信、决策过程不可篡改。

这三层组件的交互关系决定了 OpenClaw 的安全防御不能依赖单点机制，必须构建涵盖四个维度的纵深防御体系：认证边界（谁有权接入本地服务）、网络边界（哪些网络面允许被触及）、执行边界（高权限操作如何被物理隔离）以及控制边界（决策指令如何被验证）。理想情况下，这四层边界应逐层递进、互为依托：认证边界阻挡未授权访问者，网络边界缩小攻击暴露面，执行边界将突破者限制在隔离环境内，控制边界确保即使前三层失守，核心决策仍不可被劫持。

3.2 防护边界失效

尽管架构设计上具备良好的防御蓝图，但在 OpenClaw 的实际代码实现与复杂运行环境中，上述理想的纵深防御体系却往往难以维系。某一层的破坏往往会为其他边界打开缺口，使得任何单点突破都可能横向扩散为系统性失控。本节将结合安全公告中披露的具体问题进行攻击路径分类，逐一分析每层防御边界上真实发生过的安全隐患。

3.2.1 认证边界：多路径认证与授权缺口

认证边界的核心职责是确认“操作主体的合法性”。为了适配多样化的交互需求，OpenClaw 设计了从设备配对到多级管理的复杂权限体系，并开放了本地 CLI、WebUI、API 调用等多种访问通道。理想状态下，每一个访问入口都应在对应入口完成权限校验，从而形成严格的认证验证控制。

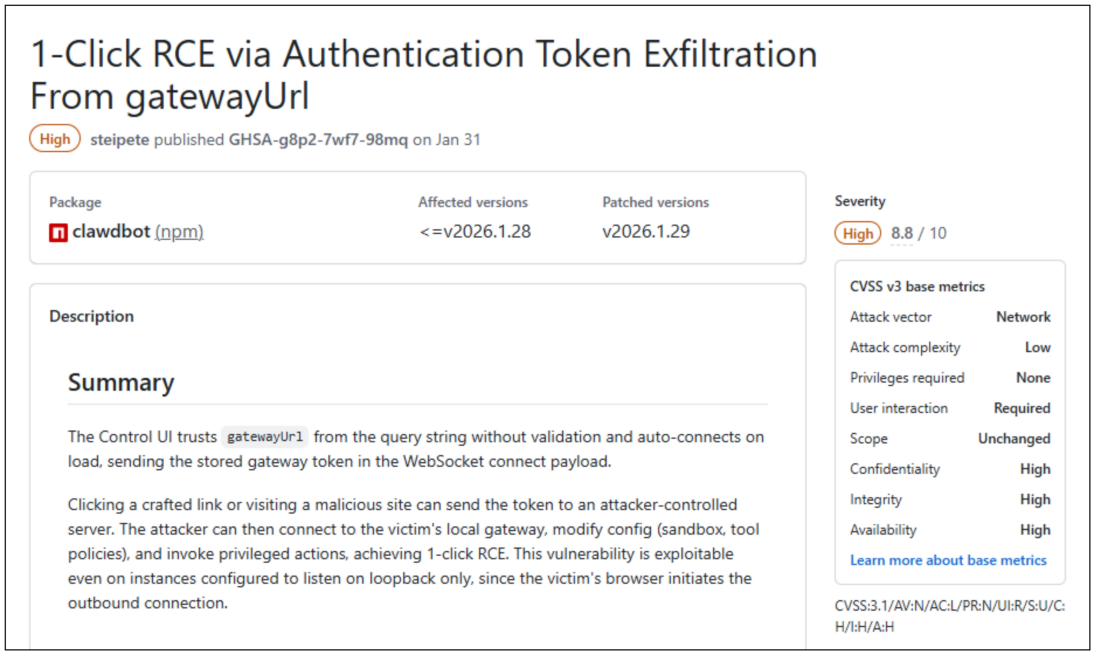


图 3-2 通过 gatewayUrl 绕过认证边界漏洞

然而，安全公告数据显示，认证与访问控制类漏洞在 OpenClaw 安全公告中占比极高。其根因在于，多层权限模型与错综复杂的访问通道交叉叠加后，容易在代码实现中产生校验盲区。攻击者可通过本地未授权接口、WebSocket 身份验证绕过、跨平台凭证复用等路径，以较低成本突破第一道防线，获得后续横向移动的机会。

3.2.2 网络边界：双向数据流转与暴露面失控

如果说认证边界管控的是访问主体，网络边界管控的则是“数据与连接的合法流向”。作为一个强交互的 Agent，OpenClaw 既需要主动外联（如抓取网页、调用第三方 API）以获取上下文，又必须提供本地服务监听以接收外部指令。

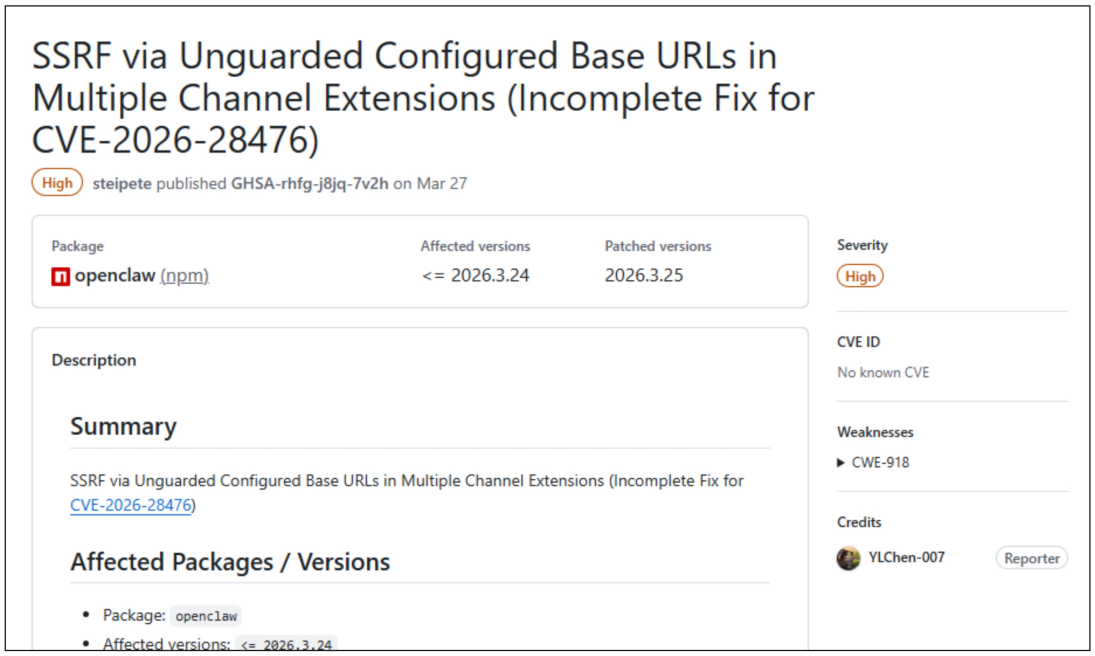


图 3-3 SSRF 突破网络边界漏洞

这种“保持高频外部感知”与“收敛自身暴露面”之间的诉求冲突，导致传统边界防御在此处趋于模糊，SSRF（服务器端请求伪造）、协议绕过等网络层风险反复出现。更为严峻的是，一旦前置的认证边界存在校验盲区，OpenClaw 内置的强大外部资源获取工具链便会成为内网探测与横向穿透的跳板工具。此时，网络边界防御不再是单纯的阻断外部恶意流量，而是演变为应对系统攻击面扩张问题。

3.2.3 执行边界：碎片化隔离与沙箱穿透

执行边界负责解决“高权限操作在物理和逻辑层面上的隔离”。OpenClaw 的设计初衷赋予了 Agent 极高的系统底层操控力，这也迫使系统必须依赖沙箱机制来限制潜在的破坏影响。这构成了执行边界最本质的挑战：既要无限逼近操作系统的核心能力，又要限制其破坏范围。

图 3-4 低权限用户提权漏洞

在实际架构中，OpenClaw 的工具调用层允许 Agent 根据任务需求自主触发外部可执行程序、浏览器实例、脚本解释器乃至容器运行时。然而，沙箱的实现却呈现出明显的碎片化特征：Docker 容器化隔离、宿主机 elevated 特权执行、以及纯逻辑策略约束三种模式并存，且按会话/代理/共享进行动态切换。不同隔离层级之间缺乏统一的强制衔接机制，导致攻击者可以在层层设防的间隙中找到逃逸路径。

安全公告中屡见不鲜的沙箱逃逸（Sandbox Escape）、本地提权（Privilege Escalation）以及通过工具链拼接实现的间接命令执行等问题表明，只要 Agent 仍具备执行底层命令的权限，任何对输入过滤的疏漏或运行环境的配置失当，都会被攻击者敏锐捕捉，将受限的工具调用转化为系统级的绝对控制权。

3.2.4 控制边界：指令可信与任务劫持

控制边界的目标是确保“用户的核心意图不被篡改”。该边界必须确保指令来源可信、决策过程不违背用户初衷，对恶意指令进行拦截。

然而，与传统软件依靠静态代码决定执行路径不同，OpenClaw 的行为轨迹完全依赖大语言模型对多源信息的非确定性推理。在这一范式下，“数据即指令”。用户输入、网页残留信息、已安装的 Skills、工具返回结果乃至文件元数据，均

可作为 Prompt 的组成部分直接影响模型决策。这种开放式的输入模式导致系统根本无法通过简单的黑白名单来拦截恶意意图。例如，作为扩展 Agent 能力的关键组件，Skill 的供应链安全直接关系到控制流的安全性。今年 2 月发生的 Claw Havoc 大规模供应链投毒事件就直接证明了这一手段已被广泛应用于针对此类 Agent 的攻击之中。

安全公告中涉及的大量提示词注入（Prompt Injection）及会话污染漏洞，揭示了一个尚未调和的架构矛盾：既要全盘接收多源输入以维持智能涌现，又要确保每一条隐式指令的绝对安全。当攻击者通过前述边界漏洞潜入系统后，Agent 自身的高级推理与自主规划能力反而会沦为攻击放大器。攻击者只需通过污染输入上下文，便能“合法地”劫持 Agent 的执行逻辑，致使所有底层的物理与网络隔离防线形同虚设。

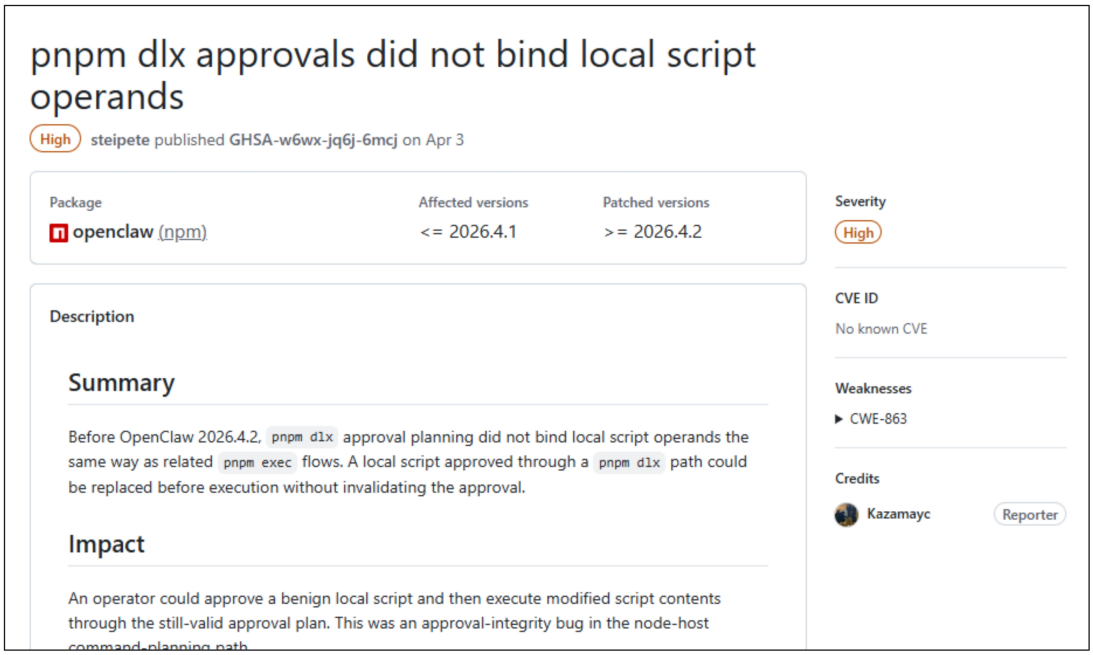


图 3-5 审批授权绕过漏洞

该授权绕过高危漏洞由 360 漏洞挖掘智能体发现。借助其内置的攻击面分析模块，智能体能够快速理解 OpenClaw 复杂的架构特征，有效构建威胁模型，并针对其薄弱环节进行深入挖掘。在对 OpenClaw 本地脚本的审批与执行流程进行剖析后，智能体发现系统仅校验了脚本的审批状态，而未对脚本内容是否被篡改进行完整性验证。攻击者可利用该缺陷，在脚本通过审批后恶意替换代码，进而在目标终端上执行非法操作，导致信息窃取、文件篡改甚至整机被控。

3.3 小结

综合上述分析，OpenClaw 的安全态势呈现出典型的“能力扩张凌驾于边界约束”的特征。短短数月内激增的 535 个安全公告，不仅是功能激进迭代下的衍生代价，更暴露出其底层架构在应对复杂威胁时的系统性脆弱。

最严峻的挑战在于防御边界的“多米诺效应”：认证、网络、执行、控制四层边界高度耦合、互为底线，任何单一维度的击穿都会引发防护体系的连锁崩塌。这种根植于架构深处的隐患，注定了常规的单点打补丁式修复难以从根本上解决安全问题。更为深远的影响是，随着 OpenClaw 作为核心技术基座被广泛落地，这些原生架构缺陷正作为一种沉重的安全债务，不可避免地向整个下游生态蔓延。无论是通过代码继承直接复用，还是进行自研，Claw 生态的衍生产品都面临着隐患被继承与扩散的系统性风险。

四、 二次开发中的安全债传递

技术的普惠往往伴随着使用门槛的降低。OpenClaw 虽具备强大的原生能力，但其复杂的配置项一度将受众局限在了极客群体之中。为了让更多普通用户也能享受到 AI Agent 带来的便利，众多厂商和开发者对其进行二次封装，打磨出了开箱即用的类 Claw 产品。

然而，这种将复杂底层技术“黑盒化”的友好交付，也在无意中掩盖了深层的系统风险。当产品在敏捷开发的驱动下为了抢占市场而飞速狂奔时，安全防线往往被无奈地留在了原地。要厘清这种风险是如何蔓延至整个生态的，我们需要回归代码的源头：一方面，审视 OpenClaw 的开发模式是如何在不知不觉中沉淀下“安全债务”的；另一方面，深入剖析这些隐患是如何在二次开发的代码链条中被继承、叠加，最终演变为终端用户必须直面的现实威胁。

4.1 功能优先：“Vibe Coding”背后的隐形代价

OpenClaw 的开发者多次在公开采访中表示，项目的高速迭代深度依赖于“Vibe Coding”——即由大模型辅助甚至主导代码生成的开发范式。这种模式在极大地释放了生产力的同时，也刻下了先天的安全基因缺陷。当前 AI 在生成代码时，其核心关注点更多的是如何让功能跑通（实现效率与正确性），而非如何抵御恶意攻击。

在项目初创期，开发者很容易陷入一种务实的权衡：Agent 的每一项能力的增加都直接转化为用户体验的提升并带来市场关注度；而底层的安全加固，却是一项看不见、摸不着且难以量化收益的防御性工程。在功能优先的产品导向下，安全设计被自然而然地后置了。当这笔规模庞大的“技术债务”随着项目走向成熟而必须被清算时，其底层重构的成本却呈指数级增长。

4.2 二次开发的内忧外患

得益于宽松的 MIT 开源协议，开发者能够轻松基于 OpenClaw 进行二次开发。然而，在代码被广泛复制与分发的同时，源头的安全隐患也正沿着软件供应链向下游生态蔓延。

此外，代码继承并非风险扩散的唯一途径。通过对市面上 5 款典型衍生产品的调研，我们发现它们普遍采用“以 OpenClaw 为核心运行时，外围叠加产品侧新功能”的架构。这种集成方式构成了威胁的两重来源：一方面，衍生产品无可避免地背负着上游项目的历史安全债；另一方面，为追求差异化竞争而引入的新功能，往往缺乏严谨的安全审计，从而产生了新的攻击敞口。

对此，360 漏洞挖掘智能体展现出了强大的分析能力。借助内置的软件成分分析（SCA）与历史漏洞库比对，智能体能够高效定位上游遗留的未修复漏洞；同时，依托其可复用且持续进化的安全专家知识库，智能体也能敏锐捕捉新功能代码中的潜在隐患。下文将结合智能体的研究结果，对这两种风险的具体表现展开深入剖析。

（1）内忧：原生继承下的版本差异

这是最常见也是最难规避的传播模式。为了追求部署的极简，部分衍生产品选择将 OpenClaw 的核心组件以二进制形式直接打包进安装包中。在这种方案下，OpenClaw 的自身缺陷会随着发行版同步至下游产品。当 OpenClaw 项目出现安全问题后，下游开发者往往缺乏提前预警的渠道，并且也很难独立针对 OpenClaw 进行修复。只有在上游开源社区发布补丁后，再经历一轮漫长的打包、测试与推送周期，才能完成一次漏洞修复。在这个补丁时间差里运行的 Claw 产品，则会处于毫无防护的危险状态。

案例：某 Claw1 Canvas 认证绕过

某 Claw1 是一款企业级二次封装产品，其内置了 OpenClaw 组件作为核心运行时。研究发现，在当时最新版本中的 OpenClaw 组件存在未修复的 Canvas 认证绕过漏洞（ZDI-CAN-29311）。

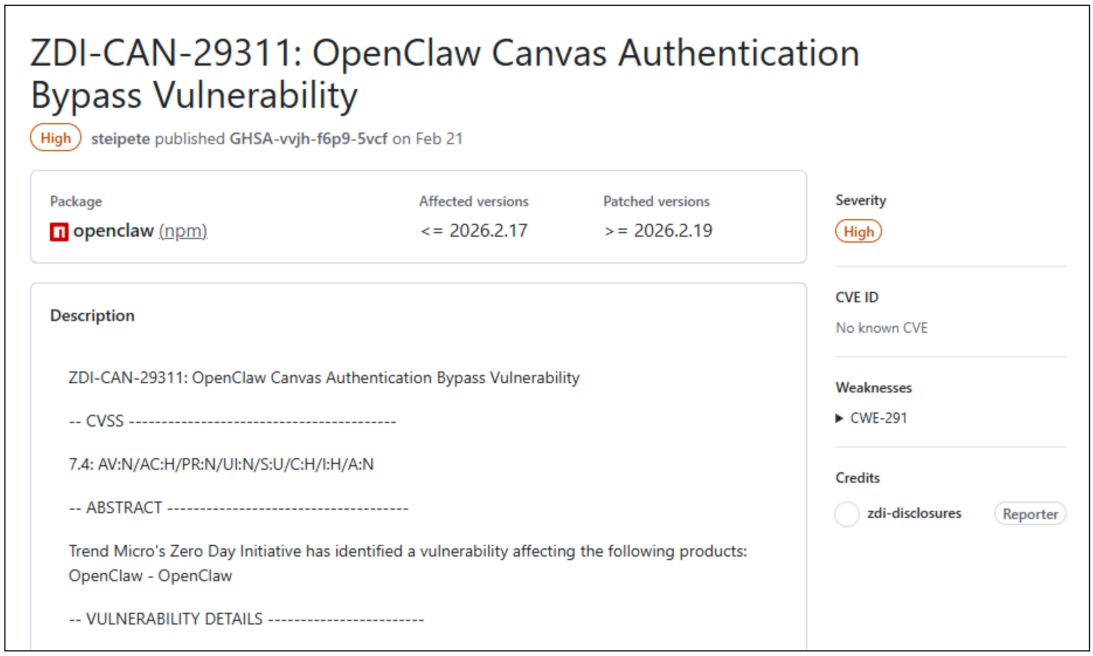


图 4-1 某 Claw1 存在 ZDI-CAN-29311 旧版本漏洞

该漏洞源于 authorizeCanvasRequest 函数中 IP 地址的认证回退机制缺陷。当 WebSocket 客户端通过私有 IP 地址认证后，所有来自该 IP 的后续 HTTP 请求都会被授予 Canvas 访问权限，而无需进行独立的身份验证。更严重的是，攻击者还可通过伪造代理头（X-Forwarded-For）绕过认证，进而以未授权的方式控制整个系统。该漏洞完全继承了 OpenClaw 原生代码中的缺陷，下游开发者在打包时未能及时更新至已修复版本，导致用户面临远程未授权访问的风险。

除了衍生产品的开发节奏与上游安全修复节奏不同步导致的漏洞延期修补问题外，如果衍生产品对 OpenClaw 进行了其它深度定制，直接升级版本还可能引入兼容性问题，这在一定程度上进一步推迟了安全补丁的应用。

（2）外患：差异化功能导致的攻击面扩张

部分封装 OpenClaw 的产品在保留核心功能的同时，为了差异化竞争而增加了新的功能模块。这些新功能往往涉及独立的网络接口、特殊文件处理逻辑或脚本执行机制，但在安全设计上可能未经充分考量，存在安全隐患。

案例 1：某 Claw2 外部页面操控浏览器

某 Claw2 在封装过程中增加了浏览器自动化操控功能（browser-agent），使 Agent 具有自主操作网页的能力。然而，该功能的中继服务在本地运行时暴露了

未鉴权的 WebSocket 端口，导致任意外部页面均可连接并操控运行中的浏览器。

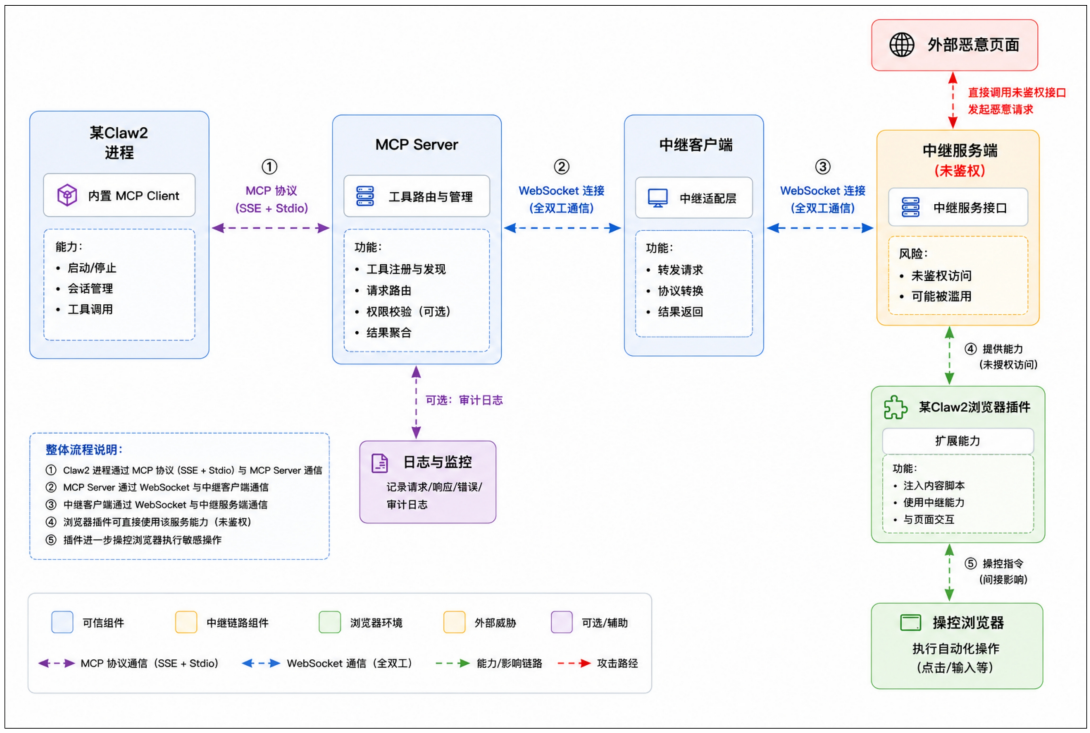


图 4-2 某 Claw2 外部恶意页面操控浏览器原理图

在此场景下，攻击者诱导用户访问恶意网页，网页通过 fetch API 连接中继服务，发送恶意指令操控某 Claw2 启动的浏览器访问特定地址 *http://127.0.0.1:<port>/get_token* ，即可窃取 Token，该 Token 后续可进一步用于相关用户认证、推理请求和文件上传等操作，导致严重的账户接管问题。该漏洞的根本原因在于，新增的浏览器控制功能未对通信通道进行适当的身份验证和来源校验，使得预期只运行在本地的服务却能由外部网页直接访问交互。

案例 2：某 Claw3 引入的邮箱与 Web Search SKILL 缺陷

某 Claw3 在 OpenClaw 的基础上额外提供更多的 Skill 以增加 Agent 能力，其邮箱服务 Skill（email-skill）中的下载附件功能未对邮箱附件名称进行过滤，攻击者可通过发送带有路径穿越文件名的恶意附件，覆盖系统关键可执行程序。

图 4-3 某 Claw3 邮箱 skill 目录穿越相关代码

此外，其内置 web_search 功能运行时默认部署在本地监听 8923 端口，该功能存在错误的 CORS 配置导致请求来源未得到有效验证，使得攻击者可通过恶意网页触发 web_search 搜索本地文件并将内容返回至攻击者。

图 4-4 某 Claw3 web_search 相关危险代码

以上案例展示了“功能引入”模式的典型特征：二次开发商在追求产品差异化的过程中，新增的功能模块往往带来新的攻击面。它们与 OpenClaw 原生的网络、认证等边界相互交叠，形成了更复杂的利用链路。

4.3 小结

OpenClaw 的安全隐患在向生态下游扩散的过程中，表现出明显的风险叠加效应。底层核心组件直接携带的历史漏洞，与外围新增业务逻辑暴露的脆弱接口，在实际运行环境中往往交叉存在并相互作用。这种内部缺陷与外部敞口的结合，导致衍生产品面临的攻击路径远比原生项目更加复杂。

这一现状揭示了当前生态安全治理面临的实际困境。在底层基础设施的遗留问题难以短期内彻底修复的客观前提下，单纯依赖下游开发者的防范意识已经无法有效阻断风险的蔓延。要真正切断这种系统性的威胁扩散，必须跨越仅靠修补代码的传统思路，转而从底层架构出发，构建即使局部组件或特定功能失陷，依然能够有效运转的系统级防御体系。

五、 开源自研的安全挑战

区别于直接基于原生框架封装的二次开发，部分开发者选择吸收 OpenClaw 的核心架构理念进行独立重构。这类项目通常会精简冗余模块，针对特定业务进行功能定制，并试图在重写过程中强化底层防护。从代码溯源的角度看，这种完全独立的重写路径确实切断了上游历史漏洞的直接传播链条。

然而，风险并未因此终结。**我们使用 360 漏洞挖掘智能体对多款独立开发的产品进行了系统性安全审计。借助该智能体的语义级代码理解、跨文件数据流追踪与 AI 驱动的逻辑推理能力，成功发现了这些产品中的安全问题。**

审计结果表明，脱离原生代码库并未真正消除系统性风险，这些产品面临的安全挑战主要集中在两个层面：一是底层设计范式趋同导致脆弱性模式的重现；二是为弥补原生缺陷而新增防御机制，却因全局设计假设不严密或边界代码实现存在偏差，转而成为非预期的新攻击入口。

5.1 设计范式层面的脆弱性延续

开源自研产品在核心定位上依然遵循了与 OpenClaw 相同的基础设定，即强调本地执行优先、维持复杂的网络通信、具备深度的系统资源操控能力以及支持多渠道实时交互。这种底层的架构要求决定了系统必须处理极为复杂的权限划分与数据流转逻辑。

当开发者将这一套复杂的运行逻辑复刻到全新的代码库时，与之相伴的防御边界冲突也会不可避免地随之迁移。在 OpenClaw 中被识别出的典型漏洞模式，依然高频出现在这些经过完全独立重写的产品中。这揭示了一个关键的工程事实，即摒弃原生代码库无法规避由架构同源性引发的漏洞跨实现复现。相同的底层设计范式，最终导致了高度相似的安全防护盲区。

为了具体说明这种架构脆弱性的延续，我们以开源自研 Claw 为例进行对比分析。该项目其在核心业务逻辑的处理上，精准复现了 OpenClaw 历史上出现过的典型漏洞。

案例：

1. 外部请求处理与 SSRF 风险

Agent 在执行任务时必然需要发起外部网络请求，如何准确拦截指向内部网络的恶意 URL，是所有此类产品必须解决的共性安全问题。在 OpenClaw 的历史记录中（如 GHSA-8cp7-rp8r-mg77 等多份安全公告），因过滤逻辑未覆盖 IPv6 到 IPv4 过渡地址等特殊格式，从而导致服务器端请求伪造（SSRF）的案例极为普遍。

开源自研 Claw4 同样重蹈了这一覆辙。尽管其代码完全重写，并在请求模块中加入了针对本地和私网地址的检测逻辑，但依然遗漏了部分边界情况。攻击者仅需构造类似 http://[::ffff:127.0.0.1] 的请求，即可直接绕过现有的检测机制，使 Agent 向运行环境的本地环回接口发起探测与攻击。

```
_BLOCKED_NETWORKS = [
    ipaddress.ip_network("0.0.0.0/8"),
    ipaddress.ip_network("10.0.0.0/8"),
    ipaddress.ip_network("100.64.0.0/10"),    # carrier-grade NAT
    ipaddress.ip_network("127.0.0.0/8"),
    ipaddress.ip_network("169.254.0.0/16"),    # link-local / cloud metadata
    ipaddress.ip_network("172.16.0.0/12"),
    ipaddress.ip_network("192.168.0.0/16"),
    ipaddress.ip_network("::1/128"),
    ipaddress.ip_network("fc00::/7"),          # unique local
    ipaddress.ip_network("fe80::/10"),         # link-local v6
]

_URL_RE = re.compile(r"https?://[^\s\"'`;|<>]+", re.IGNORECASE)


def _is_private(addr: ipaddress.IPv4Address | ipaddress.IPv6Address) → bool:
    return any(addr in net for net in _BLOCKED_NETWORKS)
```

图 5-1 开源自研 Claw4 检测请求 IP 相关代码实现

2. 外部消息流转与路径穿越隐患

支持多种即时通讯平台的接入已成为当前 Claw 类产品的标准配置。然而，复杂多样的外部消息载体极大地增加了输入处理的安全难度。OpenClaw 曾因未对外部请求中的 media 类型文件名进行过滤，导致路径穿越，产生任意文件读取漏洞（编号 GHSA-f6pf-4gjx-c94r）。

在开源自研 Claw5 处理飞书平台用户消息的独立实现中，我们检测到了类似的安全问题。系统在接收消息附件时，未对文件名执行严格的字符规范化校验，便直接进行了文件路径的拼接操作。如图 5-2 所示，攻击者通过飞书发送带有

目录跳转字符的恶意文件，即可越过 Claw5 设定的工作目录限制，将文件直接写入操作系统的更高层级目录中，而不是 Claw5 所在的工作目录。

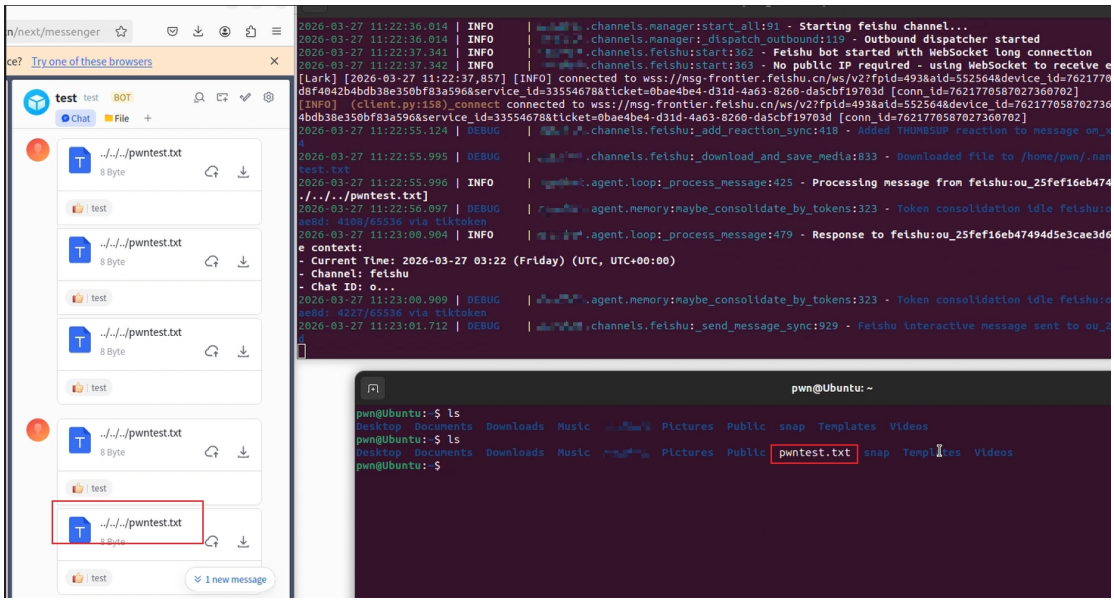


图 5-2 开源自研 Claw5 飞书实现路径穿越展示

由此可见，缺乏系统性防御视角的代码重构，最终只会让相同的安全漏洞在不同的产品载体中反复重现。

5.2 新安全机制的逻辑缺口

基于对原生架构历史漏洞的认知，部分开源自研产品在重写代码时，针对关键防御边界引入了定制化的安全加固机制。这种改进的初衷是修补已知的底层防线。然而，系统安全工程的基本规律表明，任何安全机制的叠加都会不可避免地增加代码的整体复杂性。

在缺乏全局威胁建模的敏捷开发过程中，这些新增的防御逻辑往往只关注单一的攻击维度。开发者为防范特定威胁而编写的拦截规则，极易在与系统其他基础功能（如网络解析、文件操作）的交互中产生设计断层。这导致原本用于提升安全基线的机制，反而因自身的实现偏差成为了全新的攻击入口。

案例 1：认证机制强化引发的防线穿透

开源 Claw6 是一款使用 Rust 编写，主打轻量跨平台的自研 AI 助理。为了规避原生架构在认证环节的脆弱性，其开发者采取了极简的接口暴露策略，并引入了基于六位数字配对码的动态凭证生成机制。

Claw6 部署于公网时，会暴露 3 个无需认证的 API 端点，其中/api/pair 的作用是根据 Claw6 启动时生成的配对码来生成后续的请求认证 Token，该配对码范围在[0, 1000000)，存在被未授权请求恶意爆破的可能。考虑到配对码空间较小存在被穷举的风险，系统在服务端叠加部署了防爆破策略，会对请求过于频繁的客户端进行拦截锁定。

深入研究发现，服务端识别独立客户端的唯一依据，仅仅是 HTTP 请求头中的 X-Forwarded-For 字段，而并未结合底层的真实 Socket IP 进行联合校验。这种浅层的身份识别逻辑使得攻击者可通过伪造 HTTP 请求头部模拟多个客户端进行请求从而绕过爆破防御机制，轻松耗尽配对码的计算空间，从而获取管理者 Token，并能够篡改配置或执行命令。

下图展示了绕过频率限制，在可接受的时间开销内通过爆破获 Token 值：

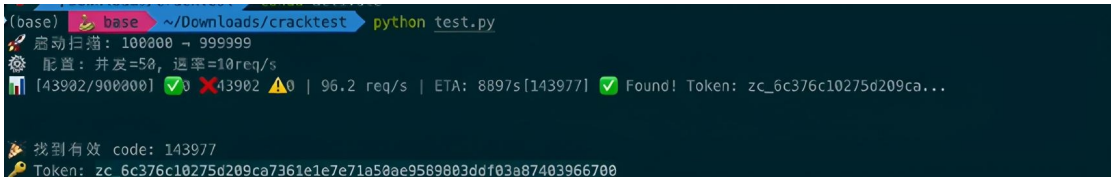


图 5-3 Claw6 Token 值爆破

在获取 Token 后，可以利用该 Token 成功访问 WebUI 页面：

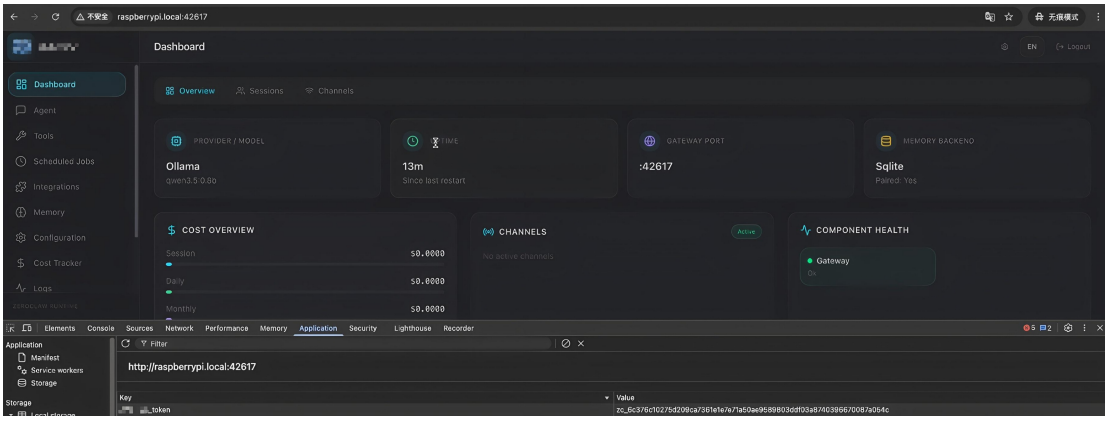


图 5-4 Claw6 通过 Token 未授权访问 WebUI

值得注意的是，由于现代浏览器引擎在底层严格限制了前端网页脚本篡改特定网络请求头的能力，此类攻击主要对直接暴露在公网环境下的服务端点构成实质威胁。

案例 2：控制边界扫描机制的绕过

开源 Claw7 是一款基于特定大模型生态开发的个人助理工具。鉴于此前频发的三方扩展组件供应链安全事件，其开发者针对控制边界引入了专门的安全扫描器。该机制会在任何外部组件被激活调用前，对其执行环境和依赖脚本进行强制静态审计，以防止恶意指令干扰核心业务逻辑。

问题在于，系统局部的安全强化并未与全局的功能设计保持步调一致。Claw7 系统内同时保留了一个用于恢复工作区数据的压缩包上传接口。该业务接口在处理文件解压时，仅对非法的目录跳转符进行了基础过滤，却没有对文件内容的深度审计逻辑。攻击者可以预先构造包含恶意执行脚本的 Zip 压缩包，通过数据恢复接口合法上传，覆盖已激活 Skill 目录下的 scripts/run.py 文件，从而绕过 SkillScanner 安全扫描机制，并在 Skill 被调用时执行任意代码。

攻击流程图如下所示：

图 5-5 Claw7 恶意 SKILL 覆盖实现 RCE 流程图

5.3 小结

开源自研绝非底层安全隐患的终结，而是风险形态的隐性转化。一方面，缺乏对核心设计架构的反思，单纯的代码重写根本无法阻断固有脆弱性的跨项目蔓延；另一方面，脱离全局视角的局部防御加固，反而会因代码交互逻辑的复杂化催生出全新的攻击敞口。

因此，针对单一节点的零散修补与功能裁剪，无法实现生态级别的安全进化。当前自研产品所面临的挑战，本质上并不是代码维度的实现缺陷，而是上游经典架构设计基因的被动延续。如果安全机制始终被设定为功能扩展后的附属限制，而不是不可妥协的前置基础，那么任何层面的技术重构都只能停留在表象。

六、 总结

OpenClaw 生态的爆发式演进，是 AI Agent 技术从理论构想走向工程落地的一个经典缩影。无论单一底层框架的市场关注度如何演变，相关的衍生产品早已深度渗透至代码开发、数据处理、终端运维等高价值场景，其使用者正迅速从专业研发人员向希望借助 AI 提升生活和工作效率的普通用户扩散。当此类高度自动化的系统广泛获取了本地文件读写、网络服务调用及关键系统命令执行权限时，其潜在的安全辐射范围已彻底跨越了个人设备的物理边界，直接对企业核心内网与关键业务流程构成实质性威胁。此次漏洞挖掘智能体的实战演练，正是对这一严峻威胁图景的真实确证。

基于自研漏洞挖掘智能体对 OpenClaw 原生架构及 10 款衍生产品的深度审计，本报告从底层架构防护失效、供应链代码继承隐患以及开源自研架构的同源性挑战三个维度，梳理当前 Claw 生态面临的真实威胁。智能体在自动化攻防中构建的复杂攻击链路表明，当前该领域面临的绝非零散的漏洞修复难题，而是“功能至上”导向下，系统性安全遗留问题的全面扩散。底层框架的固有缺陷正通过静态代码打包被下游被动继承，外围新增的业务逻辑不断撕开新的攻击敞口；即使是脱离原生代码库的自研产品，也因遵循相同的底层设计范式而在无意中复刻了同类问题。

面对这种跨越代码边界的风险蔓延，依赖事后补丁分发与被动响应的传统治理机制已然失效。生态的长期健康发展，迫切要求整个开发者完成从局部修补到系统防御的认知跨越。安全机制绝不能继续作为功能扩张后的附属约束，而必须成为产品研发的绝对基石。唯有在认证接入、网络流转、执行隔离与控制决策这四大核心边界上，严格贯彻默认安全、最小权限与纵深防御的工程准则，方能有效应对具备高自主性的 Agent 系统所带来的未知挑战。

本报告基于漏洞挖掘智能体实践所沉淀的漏洞分布形态与风险演进路径，不仅是对当前 Claw 生态的一次全面安全体检，更旨在为未来更广泛的智能化应用生态提供具有实战价值的工程参考与防御支撑。