

revagent — 黑盒程序逻辑分析 Agent 使用文档

revagent — 黑盒程序逻辑分析 Agent

基于 [pi coding agent](#) 的逆向分析 agent。你只需用自然语言和它对话——"这个程序在干什么？""帮我找出 ./crackme 的注册码""这个 APK 有没有偷偷上传数据？"——模型会自动驱动整套逆向工具链：识别文件、解包、反汇编、用 Ghidra 反编译、用 angr 求解，然后写出逻辑分析报告。你不需要手动敲任何工具命令。

支持 apk、.o、.hex、exe、ELF、PE、Mach-O、裸固件镜像，并能自动按环境装配工具链。仅做静态分析，不运行样本。

怎么用：直接对话

`revagent` 这个命令只在启动会话时用一次，之后就是聊天。

```
revagent ./suspicious.apk      # 启动会话（可顺便丢进一个文件）
revagent                       # 或空启动，在对话里说文件路径
```

然后用中文/英文直接问，agent 会自动调用工具：

```
你 ▶ 这个 apk 在偷偷联网吗？
revagent ▶ （自动 triage → unpack[apktool+jadx] → strings_scan → 读 manifest）
           它申请了 INTERNET + READ_CONTACTS。启动 Activity 的 onCreate 拉起
           UploadService，用 OkHttpClient 把通讯录 POST 到 https://api.evil.example/c2。
           证据：sources/com/app/UploadService.java:42 ...

你 ▶ ./crackme 的注册码是多少？
revagent ▶ （自动 triage → decompile[Ghidra] check() → symexec[angr]）
           该函数逐字节比较 5 个字符；angr 求解出输入："rev42"。
```

也可以完全放手："分析 ./firmware.hex，告诉我这设备是干嘛的"——它会识别、把 Intel-HEX 转成二进制、从中断向量表推断 CPU、按正确加载基址反汇编，并给出逻辑说明。

如果缺某个工具，它会先用现有工具尽量分析，再请求安装缺失工具（`provision` 工具）后继续——无需你手动配置环境。

也支持一次性非交互模式：`revagent -p "分析 ./crackme 并找出口令" ./crackme`

案例：分析汽车供应商交付的黑盒代码

背景：某 Tier-1 供应商只交付了 ECU 固件，不给源码。作为 OEM/集成方，你需要在不接触源码 的情况下：①搞清楚控制逻辑；②做安全性评估（ISO/SAE 21434 视角：有无后门、硬编码密钥、不安全内存操作、可被滥用的诊断服务）；③学习其算法（如控制算法、校验和、UDS 安全访问的 seed-key 算法）。交付物通常是 `ecu_app.hex`（Intel-HEX 固件）、`.bin` 镜像，或某个 `.o` / `.a`、甚至一个 Windows 标定工具 `.exe`。

前提：这是你被授权评估的、交付给你的产物——供应商代码安全评估属于正当逆向场景。

启动并对话：

```
revagent ./ecu_app.hex
```

你 ▶ 这是供应商交付的 ECU 固件。先确认它是什么、跑在什么芯片上，再给我一个分析计划。

revagent ▶ (triage → unpack[ihex→bin])

Intel-HEX 固件，加载基址 0x08000000、约 192KB。首字为初始 SP(0x20005000)、次字为复位向量(0x08000ab1, 奇地址=Thumb)→ ARM Cortex-M。计划：定位向量表/复位入口→主循环与 CAN 分发→标定区/版本串→重点函数反编译→安全访问算法。

你 ▶ ① 先讲控制逻辑：主循环和它怎么处理 CAN 报文。

revagent ▶ (disasm 复位入口 → 跟 bl 调用 → decompile 主循环/分发函数)

复位后初始化时钟与外设，进入 main loop。0x0800x x 处是 CAN 接收中断处理：按报文 ID 查跳转表分发—0x7E0 进入 UDS 诊断处理，0x120/0x121 是扭矩/转速控制帧。控制算法在 ctrl_step()：读传感器→PID(Kp/Ki/Kd 取自标定区 0x0801F000)→限幅→写 PWM。

你 ▶ ② 安全性评估：有没有硬编码密钥、后门、不安全内存操作、危险的诊断服务？

revagent ▶ (strings_scan + symbols + decompile 诊断处理 + 审查内存操作)

发现：

- UDS 0x27 安全访问已实现；seed→key 用固定常量异或+移位（弱算法，见③）。
- 标定区存在 16 字节疑似硬编码密钥 0x0801F100；被 routineControl 0x31 引用。
- diag 处理里有未做边界检查的 memcpy（请求长度直接取自报文，栈溢出风险）。
- 存在一条未公开的诊断子功能(0x3E 之外)，会跳过认证直接写 RAM—疑似后门。

以上均标注了地址与反编译片段作为证据。

你 ▶ ③ 学习算法：把 seed-key 安全访问算法还原出来，并验证。

revagent ▶ (decompile compute_key() → 用 symexec/重写复现)

compute_key(seed) = ((seed ^ 0xA5A5A5A5) <<< 7) + 0x3C2F。已还原为可执行伪码，并对样例 seed 校验：seed=0x11223344 → key=0x...（与固件比较逻辑一致）。

你 ▶ 把以上整理成一份评估报告。

revagent ▶ 输出：标识与平台 / 控制逻辑 / 安全发现（按风险排序，附地址证据） / 还原的算法 / 建议（删除后门子功能、加 memcpy 长度校验、更换为强 seed-key 算法） / 待确认项。

要点：你全程只是用中文提问，识别、解包、反汇编、Ghidra 反编译、angr 求解都是 agent 自己完成的；每个结论都附带地址/符号/反编译片段作为证据。若交付物是 `.exe` 标定工具或 `.o` 目标文件，换成对应路径即可，agent 会自动走 PE / 目标文件的分析流程。

模型配置（pi 的 provider）

revagent 用 pi 当前配置的模型，不写死任何模型。配置在 `~/.pi/agent/settings.json`，凭据在 `~/.pi/agent/auth.json`。完整的 provider 列表与说明见官方文档：

<https://pi.dev/docs/latest/providers>

方式一： `/login`（推荐）

启动会话后，在输入框直接输入斜杠命令：

```
revagent      # 启动
/login        # 选择 provider 登录 (Claude Pro/Max、ChatGPT/Codex、
              # GitHub Copilot 订阅，或填入各家 API Key)
/model        # 切换当前会话使用的模型
/logout       # 退出登录
```

OAuth 令牌 / API Key 会保存在 `~/.pi/agent/auth.json`，并自动续期。

方式二：环境变量 API Key

```
export ANTHROPIC_API_KEY=sk-ant-...    # 或 OPENAI_API_KEY / GEMINI_API_KEY /
    OPENROUTER_API_KEY ...
revagent ./target.exe
```

各 provider 对应的环境变量名见上面的官方 providers 文档。

方式三：设默认模型（`~/.pi/agent/settings.json`）

```
{
  "defaultProvider": "anthropic",
  "defaultModel": "claude-opus-4-8",    // 逆向任务建议用推理能力强的模型
  "defaultThinkingLevel": "high"        // off | minimal | low | medium | high | xhigh
}
```

临时覆盖（不改配置）

启动器会把 pi 的参数原样透传：

```
revagent --provider anthropic --model claude-opus-4-8 ./target.exe
revagent --model claude-sonnet-4-6:high ./target.exe      # provider/型号:thinking
revagent --list-models                                    # 列出可用模型 id
```

逆向（尤其是反编译结果的推理）对模型能力比较敏感，建议用各家较强的模型。

Agent 能做什么（它替你调用的工具）

这些工具你不用手动调，模型会自己调；它们会自动选择当前已安装的最佳后端，缺失时也会如实告知并降级处理。

工具	用途
triage	识别格式/架构/加壳，给出分析方案（一律先跑这个）
unpack	apk (apktool+jadx) · zip/7z · UPX 脱壳 · 固件分离 (binwalk) · Intel-HEX→bin
strings_scan	提取并分类字符串 (url/ip/加密常量/路径/敏感词)
symbols	导入/导出/符号/重定位/段
disasm	反汇编函数或地址区间 (rizin → objdump → capstone)
decompile	Ghidra headless 反编译伪 C (无 Ghidra 时回退 rizin)
symexec	angr 符号执行——求解能到达指定地址的输入 (注册码/口令校验)
provision	按需安装/修复工具链

它还有 pi 内置的 `bash / read / write`，并按文件类型自动加载对应技能 (triage 路由、ELF/o、PE/exe、Android apk、固件/Intel-HEX)。会话内输入 `/revagent-tools` 可查看已检测到的工具链。

自带的工具链

阶段	工具
识别 / 解包	file、binwalk、7z、unzip、upx、apktool、jadx
结构 / 反汇编	rizin / radare2、binutils、capstone
解析	lief、pyelftools、r2pipe、intelhex、unicorn、macholib
反编译	Ghidra headless (自动配套 JDK)
符号执行	angr

安装器会探测系统的发行版/包管理器，只安装缺失项，对重型工具 (Ghidra/angr) 按可用磁盘做门控，并写出能力清单 `~/.pi/agent/revagent/tools.json` 供 agent 读取。

安装

方式一：交给你的 coding agent 安装（推荐）

把整个 `revagent/` 目录发给你的编码助手（Claude Code、pi、Codex、Cursor 等），贴这段话：

这是一个逆向分析 agent 的项目。请阅读 `revagent/README.md`，然后运行 `revagent/install.sh` 帮我把工具链装好。安装器是幂等的、会自动按系统环境只装缺失的工具，并会先检测有没有 pi——没有就自动安装。装完告诉我怎么用。

agent 会读懂文档、按需执行安装、并处理过程中的环境差异（包管理器、磁盘、JDK 等）。

方式二：手动安装

```
cd revagent
./install.sh           # 完整：静态分析 + Ghidra + angr，并按需自动装 pi
./install.sh --light   # 仅静态，跳过 Ghidra + angr（磁盘紧张时用）
./install.sh --no-pi   # 不自动安装 pi（只装逆向工具链）
```

部署到 `~/pi/agent/revagent/`，并把 `revagent` 软链到 `/usr/local/bin`。安装器幂等——随时重跑可补齐缺失工具；apt 以非交互方式运行（不弹窗、不卡死）。

安装器会先检测有没有 pi：已装则跳过；未装则自动安装——优先官方安装器 `curl -fsSL https://pi.dev/install.sh | sh`，失败再回退到 `npm install -g @earendil-works/pi-coding-agent`（`--no-pi` 可关闭此行为）。pi 仅用于对话式启动器 `revagent`；工具链与脚本本身不依赖 pi——见下节。

不止用于 pi：通用性

这套东西核心是「逆向工具链 + 一组独立脚本 + 技能/提示词」，pi 只是其中一种对话外壳：

组件	是否依赖 pi	其他 agent / 人怎么用
工具链 (radare2/Ghidra/angr/jadx...)	否	系统级安装, 谁都能用
agent/bin/*.py 辅助脚本	否	任何 agent 用 <code>bash</code> 直接调用, 或人手动跑
agent/skills/*/SKILL.md	否	遵循 Agent Skills 标准 , Claude Code / Codex 等可直接加载
agent/system.md 提示词	否	作为任意 agent 的系统提示词
agent/extensions/index.ts 扩展	是	pi 专用 (封装成 8 个工具); 别的 agent 直接调脚本即可
bin/revagent 启动器	是	pi 专用

为此仓库自带跨工具适配: 根目录的 `AGENTS.md` (pi / Codex 通用) 和 `CLAUDE.md` (Claude Code) 会让任意 agent 一进仓库就知道怎么用这套工具; 安装器还会把 `agent/skills/` 自动软链到 `~/.claude/skills/` 和 `~/.codex/skills/` (若检测到对应 harness), 让 Claude Code / Codex 原生加载这些技能。

所以: 在 **Claude Code / Codex** 里, 把 `agent/skills/` 当技能加载、把 `agent/system.md` 当系统提示词, 让它用 `bash` 调 `agent/bin/` 下的脚本, 就能得到同样的能力——无需 pi。例如:

```
python3 ~/.pi/agent/revagent/bin/triage.py          ./ecu_app.hex
python3 ~/.pi/agent/revagent/bin/strings_scan.py    ./firmware.bin
python3 ~/.pi/agent/revagent/bin/hex2bin.py         ./ecu_app.hex
python3 ~/.pi/agent/revagent/bin/symexec_find.py    ./crackme --find 0x... --stdin 16
# 反汇编/反编译直接用系统里的 radare2 / Ghidra headless / objdump
```

目录结构

install.sh	环境感知安装器 (同时自部署, 供 provision 复用)
bin/revagent	启动器 (用 RE 提示词 + 扩展 + 技能 包装 pi)
agent/system.md	逆向专家系统提示词
agent/extensions/index.ts	pi 工具 (triage/unpack/.../symexec/provision)
agent/bin/*.py	辅助脚本 (triage、strings、hex2bin、capstone、ghidra、angr)
agent/skills/<格式>/SKILL.md	各格式分析 workflow
test/load_extension.mjs	扩展离线自检

安全与边界

仅用于授权分析（恶意软件研究、CTF、审计、你自己的程序）。Agent 只重建并解释程序逻辑，不产出可直接使用的恶意软件、武器化漏洞利用或免杀工具；也绝不运行目标样本——`symexec` 是在 `angr` 的隔离虚拟机里对代码路径做推理。