5月15日

recon_agennt

给定target_ip及侦察主题-->初始化侦察代理-->侦察迭代-->结果输出

初始化侦察代理

加载持久化状态：从`recon_agent_state.json`读取`assistant_id`和`thread_map`，若不存在则会创建新的

AI助手

侦察过程

运行线程：调用DeepSeek API，等待助手生成响应（如建议执行的侦察命令）

解析响应:若响应不是合法JSON，要求助手重发

执行命令：执行`json_msg['executable']`中的命令（如`nmap 192.168.1.1`），并将结果反馈给助手

终止条件：达到最大迭代次数或收到`executable`为`None`

planning_agent

配置目标-->第一轮搜索-->第二轮搜索-->结果分析与摘要生成

配置目标

提供由recon_agent侦察到的服务名及版本号信息

第一轮搜索：CVE漏洞扫描

执行`cvemap`命令扫描目标产品的CVE漏洞，并将结果保存为json文件

第二轮搜索：CVE到漏洞利用代码（PoC）

目标：基于第一轮的CVE列表，搜索公开的漏洞利用代码（PoC）

处理:由于第一轮基于产品名称搜索，找出的CVE比较多，通过AI模型分析Google搜索结果，提取最多5个关键CVE，针对筛选后的CVE，在GitHub和Google上搜索PoC

```
第一轮输出（cvemap.json）: [CVE-2023-1234, CVE-2023-5678, ...]
模型筛选后（selected_cves）: [CVE-2023-1234, CVE-2023-5678]
过滤后（filtered_cves）: [CVE-2023-1234, CVE-2023-5678]
最终搜索关键词: "CVE-2023-1234 exploit poc", "CVE-2023-5678 exploit poc"
```

```
第一轮搜索（产品 → CVE）
  ↓
生成CVE列表（cvemap_res）
  ↓
Google搜索（关键词：产品 + exploit poc）
  ↓
模型分析 → 生成关键CVE列表（selected_cves）
  ↓
过滤CVE（仅保留与cvemap_res匹配的CVE）
  ↓
第二轮搜索（CVE → GitHub/Google）
```

execute_agent

初始漏洞分析-->分步攻击指令生成-->命令执行与反馈循环

初始漏洞分析

发送初始提示（`PentestAgentPrompt.execution_init_exploit_analysis`）给AI模型，生成漏洞分析报告

将分析结果发送给侦察代理（`ReconAgent`）以获取目标信息（如IP、端口）

分步攻击指令生成

根据侦察结果的信息，生成具体的攻击指令

使用`chat_with_tool`方法调用AI模型生成分步执行命令

命令执行与反馈循环

解析AI生成的JSON指令，提取`executable`字段的命令

在指定目录（`doc_dir`）下执行命令，并将结果反馈给AI模型

循环执行直到达到`EXEC_LIMIT`或命令返回`None`

```
AI生成命令 → 执行命令 → 反馈结果 → AI调整命令 → 循环
```