Visão geral rápida

O script é um *anti-ransomware* que combina um honeypot (diretório com arquivos-isca) + monitoramento reforçado de pastas reais. Tem três comandos principais via CLI:

- **`init` — cria o honeypot (arquivos isca) e gera um índice com hashes;**
- **`monitor` — roda o monitor em tempo real (watchdog) sobre o honeypot e opcionalmente pastas reais; responde (suspende/mata) processos suspeitos se configurado;**
- **`simulate` — cria eventos inofensivos no honeypot para testar alertas (rename, append, e criação de ransom notes falsas).**

Requisitos: `pip install watchdog psutil` (psutil é opcional — sem ele o correlação por processo e ações automáticas não estarão disponíveis).

CLI — comandos e flags

1) `init`

Cria o honeypot (estrutura de diretórios, arquivos isca), indexa os arquivos (sha256) e grava um índice.

Argumentos:

- **`--dir PATH` (obrigatório): diretório base do honeypot (ex: `/opt/honeypot`).**
- **`--count INT` (default `80`): quantos arquivos-isca gerar.**

- `--subdirs INT` (default `6`): quantas subpastas criar (ex: `docs_00`, `docs_01`, ...).

- `--min-size INT` (default `4000`): tamanho mínimo em bytes dos arquivos gerados.

- `--max-size INT` (default `120000`): tamanho máximo em bytes dos arquivos gerados.

O que faz: cria arquivos com conteúdo inofensivo (aleatório), alguns arquivos temporários como `~$nome` ou `.$nome`, gera um `honeypot_index.json` em `_index/` com `path`, `sha256`, `size`, e escreve `_honeypot.log` com registro.

---

2) `monitor`

Roda o observador (watchdog) e o handler reforçado. Monitora o honeypot base (sempre) e opcionalmente pastas reais informadas com `--watch`.

Argumentos:

- `--dir PATH` (obrigatório): diretório base do honeypot (onde está o index e onde os arquivos-isca estão).

- `--threshold INT` (default `12`): limiar da pontuação acumulada para disparar alerta.

- `--window INT` (default `10`): janela em segundos para somar pontos (ScoreWindow).

- `--recursive` (flag): se incluído, o monitoramento de pastas reais (flags `--watch`) será recursivo; caso contrário, apenas topo.

- `--auto-suspend` (flag): quando disparar alerta, tenta `proc.suspend()` nos PIDs suspeitos (requer psutil e privilégios).

- `--auto-kill` (flag): quando disparar alerta, tenta `proc.kill()` nos PIDs suspeitos (mais agressivo).
- `--rehash-interval INT` (default `30`): reavalia integridade (rehash) dos arquivos indexados a cada N segundos; `0` desliga.
- `--watch PATH` (pode repetir): pasta real para monitorar; repetir a flag permite múltiplas pastas.
- `--seed-canaries` (flag): cria um canário leve (`LEIA-ME.canary.txt`) nas pastas passadas em `--watch` (arquivo somente leitura com aviso).

O que faz: cria um `ReinforcedHandler` que:

- recebe eventos (create/modify/move/delete),
- calcula pontuação para cada evento (`_score_for`),
- acumula em `ScoreWindow` e dispara `_emit_alert` quando atingir limiar,
- correlaciona processos por `open_files` e taxa de escrita via `IOTracker` (se `psutil` disponível),
- quando alerta, grava um JSON detalhado em `_alerts/` e registra no `_honeypot.log`,
- se configurado, suspende/mata processos.

Importante: o honeypot (base dir) é sempre observado recursivamente no script (`obs.schedule(handler, base_dir, recursive=True)`).

---

3) `simulate`

Gera eventos inofensivos para testar o monitor (apenas no honeypot).

Argumentos:

- `--dir PATH` **(obrigatório): diretório base do honeypot.**
- `--burst INT` **(default** `20`**): número aproximado de arquivos a tocar/modificar.**
- `--notes` **(flag): criar arquivos de "ransom note" falsos (txt/html) para testar detecção de ransom notes.**

O que faz: escolhe aleatoriamente arquivos do índice; renomeia alguns para adicionar `.enc`**, acrescenta uma linha a outros (append), e opcionalmente cria arquivos** `README_TO_DECRYPT.txt`**, etc. Restaura parte dos renames (para ser seguro).**