

Documentação - Anti-Ransomware

EMPRESA:

RANSOMBREAK

PARTICIPANTES:

ADNAN GHAZZAOUI, KAUÊ MONTEIRO, GUILHERME GIMENEZ, ENZO PAIXÃO

1. Introdução

Este projeto implementa um sistema de anti-ransomware reforçado, baseado em honeypots e monitoramento de diretórios reais. O objetivo principal é identificar atividades suspeitas relacionadas a ransomware, como modificações massivas de arquivos, criação de notas de resgate e uso de extensões suspeitas.

2. Objetivo do Projeto

O código foi desenvolvido para proteger sistemas contra ataques de ransomware através de duas estratégias principais:

- Honeypots: criação de arquivos-isca (canários) que atraem ransomware e disparam alertas ao serem alterados.
- Monitoramento: observação em tempo real de diretórios críticos, com correlação de processos e taxa de escrita para identificar comportamentos suspeitos.

3. Funcionamento do Código

O sistema possui três comandos principais:

1. init – Cria o honeypot, arquivos-isca e um índice de integridade (hash SHA-256).
2. monitor – Monitora o honeypot e diretórios reais, detectando comportamentos anômalos.
3. simulate – Gera simulações seguras de eventos de ransomware para validar o funcionamento.

O monitoramento é feito com a biblioteca Watchdog, que observa eventos de criação, modificação, exclusão e renomeação de arquivos. Esses eventos recebem uma pontuação, e quando o limiar é ultrapassado, o sistema gera alertas.

Além disso, com o uso da biblioteca psutil, o programa correlaciona processos suspeitos, podendo suspendê-los ou finalizá-los automaticamente.

4. Estrutura do Código

O código está estruturado nos seguintes módulos principais:

- Funções utilitárias – manipulação de arquivos, hashing, manipulação de permissões.
- Honeypot – criação e indexação de arquivos isca.
- Monitoramento – detecção de eventos suspeitos em diretórios.
- Simulação – geração de eventos de teste controlados.
- Gerenciadores auxiliares – ScoreWindow, IOTracker e ResponseManager para cálculo de riscos e respostas automáticas.

5. Requisitos do Sistema

- Sistema Operacional Linux

6. Fluxo de Execução

1. O usuário inicializa o honeypot com arquivos isca.
2. O monitoramento é ativado, observando tanto o honeypot quanto diretórios reais.
3. Eventos de criação, modificação, exclusão e renomeação de arquivos são avaliados.
4. Cada evento recebe uma pontuação baseada em heurísticas (ex.: extensões suspeitas, ransom notes).
5. Se a pontuação ultrapassa o limiar, o sistema gera alertas e opcionalmente suspende ou finaliza processos suspeitos.
6. Logs e alertas são armazenados para análise posterior.

7. Conclusão

Este projeto fornece uma solução prática e extensível para detectar comportamentos típicos de ransomware, usando honeypots e monitoramento de arquivos em tempo real. Apesar de suas limitações, já permite uma camada adicional de segurança e estudo acadêmico sobre defesa contra malware.

8. Código

```
#!/usr/bin/env python3
# -- coding: utf-8 --
```

```
"""
```

Anti-ransomware reforçado: honeypots + monitoramento de pastas reais.

Comandos:

```
init    -> cria honeypot + índice (igual ao original, com melhorias)
monitor -> monitora honeypot e/ou diretórios reais
simulate -> simula eventos suspeitos apenas no honeypot (seguro)
```

Requisitos:

Compatível: Windows / Linux (inclusive Kali). Em Windows, rode como Admin para suspender/kill com mais confiabilidade.

```
"""
```

```
import argparse
import hashlib
import json
import os
import random
import string
import sys
import threading
import time
from datetime import datetime
from pathlib import Path
from collections import deque, defaultdict
```

```
# ---- dependências externas
```

```
try:
    from watchdog.observers import Observer
    from watchdog.events import FileSystemEventHandler
except Exception as e:
    print("Instale watchdog: pip install watchdog", file=sys.stderr)
    raise
```

```
try:
    import psutil
    HAVE_PSUTIL = True
except Exception:
    HAVE_PSUTIL = False
```

```
# ---- utilitários
```

```
def ts():
    return datetime.utcnow().strftime("%Y-%m-%dT%H-%M-%S.%fZ")
```

```
def ensure_dir(p: Path):
    p.mkdir(parents=True, exist_ok=True)
```

```
def sha256_file(path: Path) -> str:
    h = hashlib.sha256()
    with path.open("rb") as f:
        for chunk in iter(lambda: f.read(8192), b''):
            h.update(chunk)
    return h.hexdigest()
```

```
def write_json(path: Path, data: dict):
```

```

    ensure_dir(path.parent)
    path.write_text(json.dumps(data, indent=2, ensure_ascii=False), encoding="utf-8")

def read_json(path: Path, default=None):
    if path.exists():
        return json.loads(path.read_text(encoding="utf-8"))
    return default

def human(p: Path) -> str:
    try:
        return str(p.resolve())
    except Exception:
        return str(p)

def rand_bytes(size: int) -> bytes:
    # conteúdo inofensivo + padding aleatório
    txt = ".join(random.choices(string.ascii_letters + string.digits + " _.-;", k=min(size, 1024)))
    pad_len = max(0, size - len(txt.encode("utf-8")))
    return txt.encode("utf-8") + os.urandom(pad_len)

def to_readonly(p: Path):
    try:
        os.chmod(p, 0o400 if os.name != "nt" else 0o444)
    except Exception:
        pass

def to_writable(p: Path):
    try:
        os.chmod(p, 0o600 if os.name != "nt" else 0o666)
    except Exception:
        pass

# ---- config
INDEX_DIR = "_index"
ALERTS_DIR = "_alerts"
LOG_FILE = "_honeypot.log"
INDEX_FILE = "honeypot_index.json"

BAIT_NAMES = [
    "contas_2024.xlsx", "imposto_rascunho.docx", "contrato_confidencial.pdf",
    "clientes_backup.csv", "relatorio_financeiro.xlsm", "pix_comprovantes.zip",
    "senhas_ANTIGO.txt", "projeto_final.pptx", "fotos_evento_raw.cr2",
]
BAIT_EXTS = [".docx", ".xlsx", ".xlsm", ".pptx", ".pdf", ".txt", ".csv", ".jpg", ".png", ".zip", ".bak"]
RANSOM_NOTE_HINTS = ["README", "RECOVER", "DECRYPT", "HOW_TO", "RESTORE", "UNLOCK", "PAY", "BITCOIN", "KEY"]
SUSPECT_EXTS = {".locked", ".crypt", ".enc", ".encrypted", ".pay", ".payme", ".rip", ".cryptz", ".cry", ".aes", ".aes256"}

CANARY_SUFFIX = ".canary.txt"
CANARY_CONTENT = "Arquivo canário — não mover/editar. Toque aqui é sinal de risco.\n"

# -----
# INIT (Honeypot principal)
# -----

def cmd_init(base_dir: Path, count: int, subdirs: int, min_size: int, max_size: int):
    ensure_dir(base_dir)
    ensure_dir(base_dir / INDEX_DIR)
    ensure_dir(base_dir / ALERTS_DIR)

    subs = [base_dir / f"docs_{i:02d}" for i in range(max(1, subdirs))]
    for s in subs:
        ensure_dir(s)

    created = []
    candidates = []
    for _ in range(count):
        if random.random() < 0.55:
            name = random.choice(BAIT_NAMES)
        else:
            name = ".join(random.choices(string.ascii_lowercase, k=random.randint(6,14))) + random.choice(BAIT_EXTS)

```

```

candidates.append(name)

for name in candidates:
    d = random.choice(subs)
    p = d / name
    data = rand_bytes(random.randint(min_size, max_size))
    p.write_bytes(data)
    created.append(p)

    # canários temporários (~$, .$.)
    if random.random() < 0.18:
        (d / ("~$" + name)).write_bytes(rand_bytes(random.randint(64, 256)))
    if random.random() < 0.18:
        (d / (".$" + name)).write_bytes(rand_bytes(random.randint(64, 256)))

# index (hashes)
idx = {"generated_at": ts(), "files": []}
for p in base_dir.rglob("**"):
    if p.is_file() and INDEX_DIR not in p.parts and ALERTS_DIR not in p.parts:
        try:
            idx["files"].append({"path": str(p.relative_to(base_dir)), "sha256": sha256_file(p), "size": p.stat().st_size})
        except Exception:
            pass

write_json(base_dir / INDEX_DIR / INDEX_FILE, idx)
(base_dir / LOG_FILE).write_text(f"{ts()} INIT: {len(idx['files'])} arquivos indexados\n", encoding="utf-8")
print(f"[OK] Honeypot em: {human(base_dir)}")
print(f"[OK] Index salvo: {human(base_dir / INDEX_DIR / INDEX_FILE)}")
print(f"[OK] Total arquivos: {len(idx['files'])}")

# -----
# Monitor reforçado
# -----
class ScoreWindow:
    def __init__(self, threshold=12, window_sec=10):
        self.threshold = threshold
        self.window = window_sec
        self.q = deque() # (t, score, info)
        self.lock = threading.Lock()

    def push(self, score, info):
        now = time.time()
        with self.lock:
            self.q.append((now, score, info))
            while self.q and now - self.q[0][0] > self.window:
                self.q.popleft()

    def total(self):
        now = time.time()
        with self.lock:
            while self.q and now - self.q[0][0] > self.window:
                self.q.popleft()
            return sum(s for _, s, _ in self.q)

    def snapshot(self):
        with self.lock:
            return list(self.q)

class IOTracker:
    """Rastreamento simples de taxa de escrita por PID (bytes/s aproximado)."""
    def __init__(self, horizon=5):
        self.horizon = horizon
        self.last = {}
        self.lock = threading.Lock()

    def sample(self):
        if not HAVE_PSUTIL:
            return {}
        out = {}
        with self.lock:

```

```

for p in psutil.process_iter(attrs=["pid", "name", "io_counters"]):
    pid = p.info.get("pid")
    try:
        io = p.io_counters()
        wbytes = getattr(io, "write_bytes", 0)
    except Exception:
        wbytes = 0
    t = time.time()
    if pid in self.last:
        prev_w, prev_t = self.last[pid]
        dt = max(0.001, t - prev_t)
        out[pid] = (wbytes - prev_w) / dt
        self.last[pid] = (wbytes, t)
    return out

class ResponseManager:
    def __init__(self, auto_suspend: bool, auto_kill: bool):
        self.auto_suspend = auto_suspend
        self.auto_kill = auto_kill

    def act_on(self, suspects, log_fn):
        actions = []
        if not HAVE_PSUTIL:
            return actions
        for s in suspects:
            pid = s.get("pid")
            try:
                proc = psutil.Process(pid)
                if self.auto_suspend:
                    proc.suspend()
                    actions.append({"pid": pid, "action": "suspend"})
                if self.auto_kill:
                    proc.kill()
                    actions.append({"pid": pid, "action": "kill"})
            except Exception as e:
                log_fn(f"WARN action_failed pid={pid} err={e}")
        return actions

class ReinforcedHandler(FileSystemEventHandler):
    def __init__(self, base_dir: Path, watch_dirs: list[Path], alerts_dir: Path, log_path: Path,
                 index: dict, threshold: int, window_sec: int, auto_suspend: bool, auto_kill: bool,
                 is_honey_pot: bool):
        super().__init__()
        self.base_dir = base_dir
        self.watch_dirs = [wd.resolve() for wd in (watch_dirs or [])]
        self.alerts_dir = alerts_dir
        self.log_path = log_path
        self.index = index or {"files": []}
        self.known_paths = { (base_dir / f"path").resolve() for f in self.index.get("files", []) } if is_honey_pot else set()
        self.score = ScoreWindow(threshold=threshold, window_sec=window_sec)
        self.last_alert = 0.0
        self.cooldown = 5.0
        self.auto = ResponseManager(auto_suspend, auto_kill)
        self.recent_paths = deque(maxlen=400)
        self.touched_counts = defaultdict(int)
        self.process_hits = defaultdict(int) # contagem de eventos correlacionados por PID
        self.io_tracker = IOTracker(horizon=5) if HAVE_PSUTIL else None
        self.lock = threading.Lock()
        self.is_honey_pot = is_honey_pot

    def _log(self, line: str):
        with open(self.log_path, "a", encoding="utf-8") as f:
            f.write(f"{ts()} {line}\n")

    def _in_watch_dirs(self, p: Path) -> bool:
        try:
            rp = p.resolve()
        except Exception:
            rp = p
        for wd in self.watch_dirs:

```

```

        try:
            if str(rp).startswith(str(wd)):
                return True
        except Exception:
            pass
        return False

def _score_for(self, path: Path, event_type: str):
    name_up = path.name.upper()
    s = 1

    # ransom notes típicas
    if any(h in name_up for h in RANSOM_NOTE_HINTS) and path.suffix.lower() in (".txt", ".md", ".html", ".hta"):
        s += 8 # mais pesado
        # Tripwire imediato via retorno especial:
        return s + 100 # força alerta imediato

    # extensões suspeitas
    if event_type == "moved":
        s += 2
    if path.suffix.lower() in SUSPECT_EXTS:
        s += 5

    # tocar canário do honeypot
    try:
        if path.resolve() in self.known_paths:
            s += 6
    except Exception:
        pass

    # tocar canário leve em watch_dirs
    if path.name.endswith(CANARY_SUFFIX):
        s += 7

    # explosão por diretório
    parent = str(path.parent)
    self.touched_counts[parent] += 1
    if self.touched_counts[parent] >= 12:
        s += 5
        if self.touched_counts[parent] >= 25:
            # Tripwire: explosão severa
            s += 100

    # mais peso se acontecer em pastas reais
    if self._in_watch_dirs(path):
        s += 2

    return s

def _collect_suspects(self):
    suspects = []
    suspect_pids = set()
    if not HAVE_PSUTIL:
        return suspects

    paths_set = set(self.recent_paths)
    io_rates = self.io_tracker.sample() if self.io_tracker else {}

    for p in psutil.process_iter(attrs=["pid", "name", "exe", "username", "cmdline"]):
        pid = p.info.get("pid")
        ofiles = []
        hit = False
        try:
            ofiles = p.open_files()
        except Exception:
            ofiles = []
        for of in ofiles or []:
            try:
                fp = Path(of.path)
                if any(str(fp).startswith(x) for x in paths_set):

```

```

        hit = True
        break
    except Exception:
        pass
    # adicional: alta taxa de escrita recente
    high_io = io_rates.get(pid, 0) > 200_000 # ~200KB/s escrevendo no momento
    if hit or high_io:
        if pid not in suspect_pids:
            suspects.append(**p.info, "open_files": [getattr(f, "path", "") for f in (ofiles or [])], "io_write_bps":
io_rates.get(pid, 0))
            suspect_pids.add(pid)
        return suspects

def _emit_alert(self, reason: str):
    total = self.score.total()
    now = time.time()
    if now - self.last_alert < self.cooldown:
        return

    events = self.score.snapshot()
    suspects = self._collect_suspects()

    alert = {
        "alert_time": ts(),
        "reason": reason,
        "score_total": total,
        "window_sec": self.score.window,
        "events": [
            {"t": datetime.utcfromtimestamp(t).isoformat()+"Z", "score": s, **info}
            for (t, s, info) in events
        ],
        "suspects": suspects
    }

    actions = self.auto.act_on(suspects, self._log)
    if actions:
        alert["actions"] = actions

    alert_path = self.alerts_dir / f"alert_{datetime.utcnow().strftime("%Y%m%d_%H%M%S_%f")}.json"
    write_json(alert_path, alert)
    self._log(f"ALERT reason={reason} score={total} -> {human(alert_path)}")
    print(f"[ALERTA] Atividade suspeita! Detalhes: {human(alert_path)}")

    # endurecer canários conhecidos (ganhar tempo)
    try:
        for f in self.known_paths:
            if Path(f).is_file():
                to_readonly(Path(f))
    except Exception:
        pass

    self.last_alert = now

# ---- watchdog callbacks
def _handle_event(self, p: Path, event_type: str, extra=None):
    s = self._score_for(p, event_type)
    info = {"type": event_type, "path": str(p)}
    if extra:
        info.update(extra)
    self.score.push(max(1, s), info)
    self.recent_paths.append(str(p))
    if s >= 100: # Tripwire: alerta imediato
        self._emit_alert(f"tripwire ({event_type}): {p.name}")
    else:
        # alerta se passou limiar
        if self.score.total() >= self.score.threshold:
            self._emit_alert("comportamento cumulativo suspeito")

def on_created(self, event):
    if event.is_directory: return

```



```

    p = Path(event.src_path)
    self._log(f"CREATE {human(p)}")
    self._handle_event(p, "created")

def on_modified(self, event):
    if event.is_directory: return
    p = Path(event.src_path)
    self._log(f"MODIFY {human(p)}")
    self._handle_event(p, "modified")

def on_moved(self, event):
    if event.is_directory: return
    src = Path(event.src_path); dst = Path(event.dest_path)
    self._log(f"MOVE {human(src)} -> {human(dst)}")
    self._handle_event(dst, "moved", extra={"src": str(src), "dst": str(dst)})

def on_deleted(self, event):
    if event.is_directory: return
    p = Path(event.src_path)
    self._log(f"DELETE {human(p)}")
    self._handle_event(p, "deleted")

def seed_canaries_in_dirs(dirs: list[Path]):
    seeded = []
    for d in dirs:
        try:
            ensure_dir(d)
            cpath = d / f"LEIA-ME{CANARY_SUFFIX}"
            if not cpath.exists():
                cpath.write_text(CANARY_CONTENT, encoding="utf-8")
                to_readonly(cpath)
                seeded.append(str(cpath))
        except Exception:
            pass
    return seeded

def cmd_monitor(base_dir: Path, threshold: int, window_sec: int, recursive: bool,
                auto_suspend: bool, auto_kill: bool, rehash_interval: int,
                watch_dirs: list[Path], seed_canaries: bool):
    ensure_dir(base_dir / ALERTS_DIR)
    log_path = base_dir / LOG_FILE
    index = read_json(base_dir / INDEX_DIR / INDEX_FILE, default={"files": []})

    if seed_canaries and watch_dirs:
        planted = seed_canaries_in_dirs(watch_dirs)
        if planted:
            print(f"[INFO] Canários leves criados em {len(planted)} pastas.")

    # integridade (opcional, apenas honeypot)
    stop_flag = {"stop": False}
    def rehasher():
        while not stop_flag["stop"]:
            try:
                start = time.time()
                changed = []
                for f in index.get("files", []):
                    p = base_dir / f["path"]
                    if p.exists() and p.is_file():
                        try:
                            h = sha256_file(p)
                            if h != f["sha256"]:
                                changed.append(str(p))
                        except Exception:
                            pass
                if changed:
                    with open(log_path, "a", encoding="utf-8") as f:
                        f.write(f"{ts()} INTEGRITY_CHANGED count={len(changed)}\n")
                    spent = time.time() - start
                    time.sleep(max(1, rehash_interval - spent))
            except Exception:
                pass

```

```

        time.sleep(rehash_interval)

t = None
if rehash_interval > 0:
    t = threading.Thread(target=rehasher, daemon=True)
    t.start()

handler = ReinforcedHandler(
    base_dir=base_dir,
    watch_dirs=watch_dirs,
    alerts_dir=base_dir / ALERTS_DIR,
    log_path=log_path,
    index=index,
    threshold=threshold,
    window_sec=window_sec,
    auto_suspend=auto_suspend,
    auto_kill=auto_kill,
    is_honeypot=True
)

obs = Observer()
# 1) Honeypot base
obs.schedule(handler, str(base_dir), recursive=True)
# 2) Pastas reais (se fornecidas)
if watch_dirs:
    for wd in watch_dirs:
        obs.schedule(ReinforcedHandler(
            base_dir=base_dir,
            watch_dirs=watch_dirs,
            alerts_dir=base_dir / ALERTS_DIR,
            log_path=log_path,
            index=index, # índice serve só para canários do honeypot
            threshold=threshold,
            window_sec=window_sec,
            auto_suspend=auto_suspend,
            auto_kill=auto_kill,
            is_honeypot=False
        ), str(wd), recursive=recursive)

print(f"[INFO] Monitorando honeypot: {human(base_dir)} (rec={True})")
if watch_dirs:
    for wd in watch_dirs:
        print(f"[INFO] Monitorando pasta real: {human(wd)} (rec={recursive})")
print(f"[INFO] Limiar: score >= {threshold} na janela de {window_sec}s")
if auto_suspend: print("[INFO] Resposta: SUSPENDER processos suspeitos")
if auto_kill:    print("[INFO] Resposta: KILL (agressivo)")
print(f"[INFO] Logs: {human(log_path)} | Alertas: {human(base_dir / ALERTS_DIR)}")

obs.start()
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print("\n[INFO] Encerrando monitor...")
finally:
    stop_flag["stop"] = True
    obs.stop()
    obs.join()

# ---- SIMULATE (seguro; só no honeypot)
def cmd_simulate(base_dir: Path, burst: int, notes: bool):
    idx = read_json(base_dir / INDEX_DIR / INDEX_FILE, default={"files": []})
    files = [base_dir / f["path"] for f in idx.get("files", []) if (base_dir / f["path"]).exists()]
    if not files:
        print("Nada para simular. Rode 'init' primeiro.")
        return

    random.shuffle(files)
    picks = files[:max(1, min(burst, len(files)))]

```

```

renamed = []
for p in picks[: int(len(picks)*0.6)]:
    new = p.with_suffix(p.suffix + ".enc")
    try:
        p.rename(new)
        renamed.append((p, new))
    except Exception:
        pass

for p in picks[int(len(picks)*0.6):]:
    try:
        with open(p, "ab") as f:
            f.write(b"\n# simulacao_inofensiva\n")
    except Exception:
        pass

if notes:
    for name in ["README_TO_DECRYPT.txt", "HOW_TO_RESTORE_FILES.txt", "RECOVER_YOUR_DATA.html"]:
        try:
            (base_dir / name).write_text(
                "Simulação segura: isto NÃO é malware. Teste do detector de ransom note.",
                encoding="utf-8"
            )
        except Exception:
            pass

time.sleep(0.5)

for (old, new) in renamed[: max(1, len(renamed)//3)]:
    try:
        if new.exists():
            new.rename(old)
    except Exception:
        pass

print(f"[OK] Simulação concluída. Eventos gerados: ~{len(picks)}")

# ---- CLI
def main():
    p = argparse.ArgumentParser(description="Anti-ransomware reforçado (honeypot + monitoramento de pastas reais).")
    sub = p.add_subparsers(dest="cmd", required=True)

    p_init = sub.add_parser("init", help="criar honeypot e índice")
    p_init.add_argument("--dir", type=Path, required=True, help="diretório base do honeypot")
    p_init.add_argument("--count", type=int, default=80, help="quantidade de arquivos-isca")
    p_init.add_argument("--subdirs", type=int, default=6, help="quantidade de subpastas")
    p_init.add_argument("--min-size", type=int, default=4_000, help="tamanho mínimo dos arquivos (bytes)")
    p_init.add_argument("--max-size", type=int, default=120_000, help="tamanho máximo dos arquivos (bytes)")

    p_mon = sub.add_parser("monitor", help="monitorar e responder a atividade suspeita")
    p_mon.add_argument("--dir", type=Path, required=True, help="diretório base do honeypot")
    p_mon.add_argument("--threshold", type=int, default=12, help="limiar de alerta (pontuação)")
    p_mon.add_argument("--window", type=int, default=10, help="janela de soma de pontos (segundos)")
    p_mon.add_argument("--recursive", action="store_true", help="monitorar recursivamente as pastas reais")
    p_mon.add_argument("--auto-suspend", action="store_true", help="suspender processos suspeitos ao alertar")
    p_mon.add_argument("--auto-kill", action="store_true", help="finalizar processos suspeitos ao alertar (agressivo)")
    p_mon.add_argument("--rehash-interval", type=int, default=30, help="rehash periódico de integridade do honeypot (s (0 = desliga)")
    p_mon.add_argument("--watch", type=Path, action="append", default=[], help="pasta real para monitorar (repita a flag)")
    p_mon.add_argument("--seed-canaries", action="store_true", help="criar canários leves nas pastas de --watch")

    p_sim = sub.add_parser("simulate", help="simular atividade suspeita (inofensiva) no honeypot")
    p_sim.add_argument("--dir", type=Path, required=True, help="diretório base do honeypot")
    p_sim.add_argument("--burst", type=int, default=20, help="qtd. aproximada de arquivos a tocar")
    p_sim.add_argument("--notes", action="store_true", help="criar 'ransom notes' falsas")

    args = p.parse_args()

    if args.cmd == "init":

```

```
    cmd_init(args.dir, args.count, args.subdirs, args.min_size, args.max_size)
elif args.cmd == "monitor":
    cmd_monitor(args.dir, args.threshold, args.window, args.recursive,
                args.auto_suspend, args.auto_kill, args.rehash_interval,
                args.watch, args.seed_canaries)
elif args.cmd == "simulate":
    cmd_simulate(args.dir, args.burst, args.notes)
else:
    p.print_help()

if __name__ == "__main__":
    main()
```