Documentação Técnica Aprofundada: Porão Anti-Ransomware

1 Filosofia Central: Defesa em Profundidade Híbrida

O princípio fundamental da ferramenta é não confiar em uma única "bala de prata". A defesa é construída em camadas, combinando duas estratégias principais:

- **Vigilância Proativa**: Tenta ativamente encontrar o malware antes que ele cause dano.
- **Detecção Reativa**: Reage em tempo real a ações suspeitas que já estão ocorrendo no sistema de arquivos.

Qualquer uma dessas camadas, ao detectar uma ameaça, aciona um protocolo de resposta unificado e inteligente.

2 Componente 1: O Vigilante Proativo de Processos

2.1 O que estamos usando?

A biblioteca `psutil` dentro de um loop `while True` de alta frequência (intervalo de 0,05s) na função `novos_processos()` do arquivo `porao.py`.

2.2 Por que disso?

Ransomwares precisam ser executados para agir. Em vez de esperar que eles modifiquem arquivos, podemos pegá-los no exato momento em que são iniciados. Um loop de alta frequência, ao contrário de um evento, nos dá controle total sobre a frequência com que verificamos por novas ameaças, tornando-nos extremamente rápidos na detecção de processos. A biblioteca `psutil` é a padrão e mais eficiente em Python para listar e interrogar processos do sistema.

2.3 Qual o intuito de usar isso?

O objetivo é a prevenção. Esta é a nossa primeira e mais agressiva linha de defesa, projetada para neutralizar a ameaça no "segundo zero", antes que a primeira criptografia de arquivo ocorra.

2.4 Exemplo Prático

Um usuário baixa um arquivo `installer.exe` malicioso e o executa. O arquivo é extraído e tenta rodar um outro processo, `run.exe`, de dentro da pasta `C:\Users\User\AppData\Local\Temp`. O loop `novos_processos`, rodando 20 vezes por segundo, imediatamente detecta um novo processo (`run.exe`) cujo caminho do executável (`exe_path`) começa com uma das `FORBIDDEN_EXEC_PATHS`. Antes mesmo que `run.exe` possa ler o primeiro arquivo para criptografar, `encerrar_proctree()` é chamado e o processo é eliminado.

3 Componente 2: O Sensor Reativo de Eventos de Arquivo

3.1 O que estamos usando?

A biblioteca `watchdog` e a classe `MonitorFolder` que herda de `FileSystemEventHandler`.

3.2 Por que disso?

O `watchdog` se integra diretamente com as APIs do sistema operacional (como o `I/O Completion Ports` no Windows) para receber notificações em tempo real sobre eventos de arquivo. Isso é muito mais eficiente do que verificar manualmente os arquivos repetidamente. Ele nos diz "algo aconteceu neste exato momento", permitindo uma reação instantânea a ações que o Vigilante Proativo possa não ter pego.

3.3 Qual o intuito de usar isso?

O objetivo é ser o gatilho de resposta rápida para qualquer interação maliciosa com o sistema de arquivos. Se o ransomware não for pego na inicialização, ele será pego assim que tocar no primeiro arquivo de forma suspeita.

3.4 Exemplo Prático

Um ransomware já em execução começa a criptografar os arquivos da pasta "Documentos". Ele encontra e modifica o arquivo `dados_bancarios.xlsx`, que está na lista de `CANARY_FILES`. No exato milissegundo em que a modificação ocorre, o `watchdog` notifica o `MonitorFolder`. A função `on_modified` é executada, verifica que o arquivo modificado é um arquivo isca e chama `encerrar_proctree()` imediatamente.

4 Componente 3: O Sistema de Snapshot (A "Memória")

4.1 O que estamos usando?

Funções personalizadas (`criar_snapshot_arquivos`, `analisar_diferenca_e_agir`) e um dicionário Python (`SNAPSHOT_ARQUIVOS`) para armazenar o estado dos arquivos.

4.2 Por que disso?

Detectar o ransomware é apenas metade da batalha. A outra metade é remediar o dano. Uma simples lista de "arquivos recentes" é imprecisa. Um snapshot nos dá uma fotografia exata do "estado seguro" do sistema (caminho e data de modificação de cada arquivo).

A comparação entre o "antes" (snapshot) e o "depois" (durante o ataque) é a forma mais precisa de identificar todo o estrago (arquivos criados e modificados).

4.3 Qual o intuito de usar isso?

O objetivo é inteligência de resposta e remediação completa. Em vez de apenas colocar em quarentena o arquivo que disparou o alarme, garantimos que 100% dos arquivos afetados pelo ataque sejam identificados e contidos, transformando uma possível catástrofe em um incidente gerenciável.

4.4 Exemplo Prático

O antivírus detecta uma ameaça. A função `encerrar_proctree` chama `analisar_diferenca_e_agir`. Esta função varre o Desktop e encontra 15 novos arquivos `.wnry` que não estavam no último snapshot, além de um arquivo `trabalho.docx` cuja data de modificação é mais recente que a registrada no snapshot. O sistema identifica todos os 16 arquivos como parte do incidente e move cada um deles para a quarentena, limpando completamente a área de trabalho do dano visível.

5 Componente 4: O Protocolo de Resposta Unificado

5.1 O que estamos usando?

A função central `encerrar_proctree()`, que orquestra a resposta completa.

5.2 Por que disso?

Centralizar a resposta garante que, não importa como a ameaça foi detectada (seja por um processo suspeito ou por um arquivo modificado), a reação será sempre a mais forte e completa possível. Evita a duplicação de código e garante consistência.

5.3 Qual o intuito de usar isso?

O objetivo é eficácia e robustez. Garantir que cada alerta seja tratado com a máxima seriedade, executando a análise de danos, a neutralização de processos e a reconfiguração do sistema (novo snapshot) em uma sequência lógica e poderosa.

5.4 Exemplo Prático

Seja um processo rodando da pasta `Temp` ou um arquivo isca sendo modificado, ambos os eventos levam a uma única chamada: `encerrar_proctree()`. Esta função então executa sua sequência:

1. Chama `analisar_diferenca_e_agir` para conter o dano aos arquivos.
2. Usa `taskkill` para eliminar os processos.
3. Espera o sistema estabilizar.
4. Chama `criar_snapshot_arquivos` para preparar o sistema para o futuro.