# Arquitetura da Interface Gráfica e Lógica de Decisão Avançada

## 1. Visão Geral

Este documento detalha a arquitetura proposta para a interface gráfica (GUI) e a lógica de decisão avançada do programa de pentest automatizado. A GUI será desenvolvida em PyQt5, visando um tema "hacker dark minimalista" e fornecerá atualizações em tempo real e animações. A lógica de decisão será colaborativa, onde o programa sugere ações e o usuário confirma ou modifica.

## 2. Arquitetura da Interface Gráfica (PyQt5)

### 2.1. Tecnologia Escolhida

* **Framework:** PyQt5
* **Justificativa:** Robusto para aplicações desktop, alta customização para temas (incluindo dark/hacker), bom suporte a widgets complexos, capacidade de integração com animações (via QPainter, QPropertyAnimation, ou integração com bibliotecas de animação se necessário), e maturidade da biblioteca.

### 2.2. Estrutura da Janela Principal

A interface será composta por uma janela principal dividida em seções:

* **Painel de Controle (Superior/Lateral):**
  + Input para o(s) alvo(s).
  + Botão "Iniciar Varredura / Pentest".
  + Botão "Parar/Pausar".
  + Controles de "Velocidade do Processo" (simulando aceleração/desaceleração, talvez ajustando timeouts ou threads do backend).
  + Indicador de status geral (Ex: "Ocioso", "Varrrendo", "Explorando", "Aguardando Decisão").
* **Painel de Visualização Principal (Central):**
  + **Visualização Gráfica do Alvo:** Uma representação esquemática do sistema alvo (pode ser simples inicialmente). A animação do "vírus vermelho infectando" ocorrerá sobre esta visualização.
  + **Log de Atividades em Tempo Real:** Uma área de texto rolável exibindo as ações do programa, descobertas e status em tempo real. Similar a um console, mas integrado à GUI.
* **Painel de Resultados e Descobertas (Lateral/Inferior):**
  + Lista de hosts descobertos.
  + Para cada host, lista de portas abertas, serviços identificados e versões.
  + Lista de vulnerabilidades potenciais encontradas (com severidade).
  + Detalhes da vulnerabilidade selecionada.
* **Painel de Interação e Decisão (Pop-up/Seção Dedicada):**
  + Quando o programa precisar de uma decisão, esta área se tornará ativa.
  + Apresentará a sugestão do programa (Ex: "Serviço SSH encontrado na porta 22. Sugiro tentar brute-force com lista de senhas comum. Confirmar?").
  + Opções para o usuário: "Confirmar", "Modificar Parâmetros", "Ignorar Sugestão".

### 2.3. Tema e Estilo

* **Tema:** Hacker Dark Minimalista.
  + Cores: Fundo escuro (tons de cinza escuro, preto), texto em cores contrastantes (verde neon, ciano, branco para informações; vermelho/laranja para alertas/vulnerabilidades).
  + Fontes: Monospace ou fontes com estilo "tech".
  + Ícones: Minimalistas e temáticos.
* **Layout:** Limpo, organizado, com foco na informação essencial. Evitar excesso de elementos visuais desnecessários.

### 2.4. Animações

* **Animação Principal ("Vírus Infectando"):**
  + Ocorrerá no Painel de Visualização Principal.
  + Um ícone/efeito visual vermelho (o "vírus") se moverá em direção a uma representação do alvo.
  + Conforme portas são descobertas ou vulnerabilidades exploradas, partes da representação do alvo podem mudar de cor (para vermelho) ou exibir o ícone do vírus sobre elas.
  + Estilo: Pode ser uma animação 2D simples usando QPainter para desenhar formas e linhas que se movem e mudam de cor, ou sprites animados.
* **Outras Animações/Feedback Visual:**
  + Indicadores de progresso para varreduras.
  + Efeitos sutis ao passar o mouse sobre elementos interativos.
  + Transições suaves entre diferentes visualizações (se aplicável).

### 2.5. Fluxo de Dados com o Backend (Motor de Pentest)

* **Comunicação:** A GUI e o backend (código Python existente) rodarão em threads separadas para evitar que a interface congele durante operações demoradas.
  + **PyQt Signals and Slots:** Serão usados para comunicação assíncrona. O backend emitirá sinais com atualizações de status, resultados de varredura, descobertas de vulnerabilidades, e pedidos de decisão.
  + A GUI terá slots conectados a esses sinais para atualizar os widgets correspondentes (logs, listas de resultados, visualização gráfica).
* **Envio de Comandos:** Ações do usuário na GUI (Ex: clicar em "Iniciar Varredura") acionarão métodos no backend, possivelmente através de uma fila de comandos ou chamadas diretas a métodos de uma classe controladora do backend (que também gerenciará as threads).

## 3. Arquitetura da Lógica de Decisão Colaborativa

### 3.1. Funcionamento

O objetivo é que o programa "pense" sobre os próximos passos, mas envolva o usuário no processo de decisão, conforme solicitado.

1. **Descoberta:** O motor de pentest (backend) identifica uma porta aberta, serviço, ou vulnerabilidade potencial.
2. **Análise Interna (Sugestão):**
   * O backend analisa a descoberta.
   * Com base em regras predefinidas, heurísticas, ou até mesmo um modelo simples de pontuação, ele formula uma ou mais sugestões de próximos passos (Ex: "Porta 22/SSH aberta. Sugestão 1: Tentar login com credenciais padrão. Sugestão 2: Verificar versão do SSH para exploits conhecidos.").
   * A sugestão deve incluir o porquê da ação ser recomendada.
3. **Apresentação ao Usuário:**
   * O backend emite um sinal para a GUI com a(s) sugestão(ões).
   * A GUI exibe a(s) sugestão(ões) no Painel de Interação e Decisão, explicando o raciocínio.
4. **Decisão do Usuário:**
   * O usuário pode:
     + **Confirmar:** Aceita a sugestão principal e o backend prossegue.
     + **Escolher Alternativa:** Se múltiplas sugestões forem dadas.
     + **Modificar Parâmetros:** (Opcional, para implementações futuras mais avançadas) Permitir ao usuário ajustar parâmetros da ação sugerida (Ex: fornecer uma lista de senhas diferente para o brute-force).
     + **Ignorar/Pular:** O programa registra a decisão e pode tentar outra abordagem ou aguardar novas descobertas.
5. **Execução:** O backend executa a ação aprovada pelo usuário.

### 3.2. Componentes da Lógica de Decisão no Backend

* **Módulo de Análise de Descobertas:** Recebe dados brutos do scanner (Nmap, etc.).
* **Base de Conhecimento/Regras:** Um conjunto de regras ou heurísticas que mapeiam tipos de descobertas (serviços, portas, versões de software) para possíveis ações de exploração ou enumeração subsequentes. Inicialmente pode ser simples (Ex: if port == 22 then suggest ssh\_bruteforce, suggest ssh\_version\_check).
* **Módulo Gerador de Sugestões:** Cria as mensagens de sugestão formatadas para a GUI.
* **Módulo de Gerenciamento de Estado:** Mantém o controle do que já foi tentado para evitar loops ou ações redundantes.

### 3.3. Integração com o Motor de Pentest Existente

* O main.py atual será refatorado para se tornar o núcleo do backend.
* As classes NetworkScanner, SSHExploiter, HTTPExploiter serão chamadas por este núcleo.
* A lógica de decisão será uma nova camada que se situa entre a descoberta (pelo NetworkScanner) e a ação (pelos Exploiters).

## 4. Próximos Passos (Implementação)

1. **Configuração do Ambiente PyQt5.**
2. **Desenvolvimento do Layout Base da GUI:** Criar a janela principal e os painéis estáticos.
3. **Implementação da Comunicação Backend-GUI:** Estabelecer signals/slots para exibir logs básicos.
4. **Desenvolvimento do Módulo de Lógica de Decisão (Backend):** Implementar a geração de sugestões simples.
5. **Implementação do Painel de Interação e Decisão (GUI).**
6. **Integração da Lógica de Decisão com os Módulos de Scanner e Exploração.**
7. **Desenvolvimento das Animações e Visualização em Tempo Real.**
8. **Testes e Refinamentos Contínuos.**

Este documento de arquitetura servirá como guia para as próximas fases de implementação.