# Projeto: Captura de Consumo de Alarmes FTAE via Injeção COM

## Objetivo

O projeto visa analisar em profundidade como o **FactoryTalk Alarms and Events (FTAE)** do **FactoryTalk Studio v13** (ambiente distribuído) consome e manipula eventos de alarme.  
A ideia é **interceptar as chamadas COM** feitas pelo cliente e servidor, permitindo investigar:

* Como ocorre a **escuta** de alarmes.
* O comportamento interno ao **receber eventos**.
* Como cada alarme é **configurado e instanciado**.

## Ambiente de Teste

* **SO:** Windows 10
* **IDE:** Visual Studio 2022
* **Linguagem:** C# 7.3
* **Framework:** .NET Framework 4.8
* **Ferramentas auxiliares:** PowerShell 5.1 para execução de testes e automação.
* **Execução:** Cliente e Servidor FTAE rodando na **mesma máquina**.

## Estrutura da Solução

A solução (Banner.sln) contém dois projetos principais

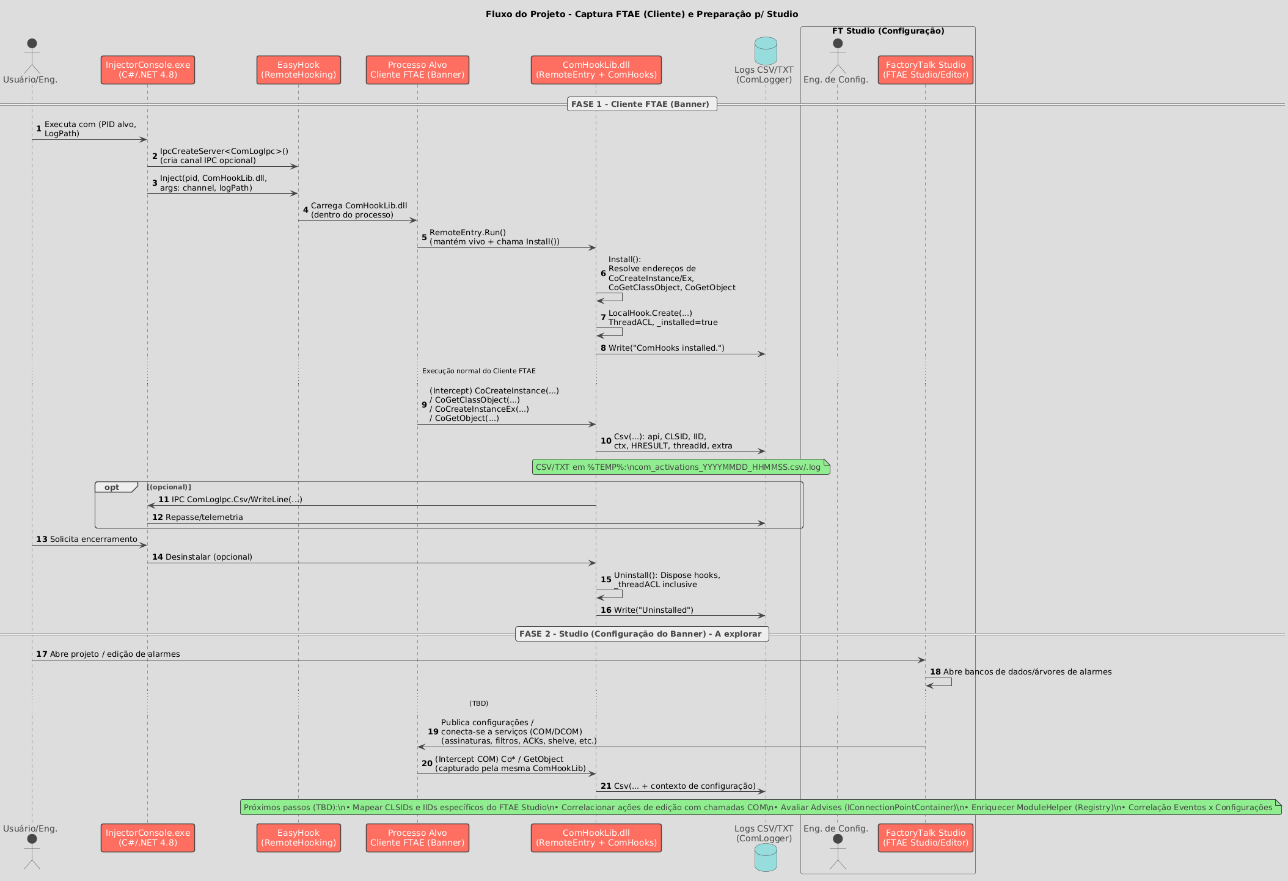


Figura 1 - Fluxo do prgrama

## Etapa 1/4 – Base de Injeção e Placeholders

* InjectorConsole implementado (Program.cs) com suporte a parâmetros PID, LogPath e DLL.
* RemoteEntry (ComHookLib) carregando hooks de UI (UiHook) e inicializando COM em modo placeholder.
* UiHook captura eventos de janela (window.create, window.show, window.foreground).
* ComHooks apenas inicializado como placeholder (sem interceptar CoCreateInstance ainda).
* Logging estruturado ativo: Jsonl, ComLogger, ComLogIpc (NamedPipe + fallback para arquivo).

Status atual:

* ✔️ Injeção confirmada
* ✔️ Hooks de UI ativos
* ✔️ Placeholder COM inicializado
* ⏸️ Hooks reais de COM ainda não instalados (aguardam etapa 2/4).

## Próxima etapa: 2/4 – Instalação dos hooks COM (CoCreateInstance/Ex).

* Instrução de Continuidade (GUI – “Adicionar Instruções”):
* Registrar no log/documentação qual etapa foi concluída, guardar no DOC ou JSON auxiliar,
* e exibir no início da GUI:
* Última etapa concluída: 1/4 – Base de Injeção e Placeholders
* Próxima etapa sugerida: 2/4 – Instalação dos hooks COM.