Project Title: Secure Language Assembly Inspector Tool (SLAIT)

Team Members:

- Maria Linkins-Nielsen (mlinkinsniel2022@my.fit.edu)
- Michael Bratcher (mbratcher2021@my.fit.edu)

Faculty Advisor: Dr. Marius Silaghi (msilaghi@fit.edu)

Client: Dr. Marius Silaghi CSE Professor & Faculty Advisor

Meeting Dates with Client:

- Meeting 1: Aug 27, 2025

Goal and Motivation: Develop a secure, streamlined way to inspect snippets of code at a low level of abstraction.

Students within the CSE 3120 “Computer Architecture and Assembly” class are required to place their systems into a vulnerable state to perform tasks involving executing “low-level language” based code (LLLC), such as those running x86 and Java Bytecode. This can leave the system at risk of unintentional damage by student actors from faulty lines of LLLC that would usually be stopped by the system, or by bad actors taking advantage of a system placed in such a state. This issue does not just affect these students, but anybody attempting to optimize their code by inspecting the assembly, as observing and running assembly to inspect registers is usually a tedious task that involves placing your PC in a vulnerable state. “SLAIT” attempts to solve this problem by code requested, allowing the user to specify registers and flags at specific locations in the code, compiling it within a Docker environment, and returning the requested information to the user, all without the user being required to lower security levels within their device.

Approach (key features of the system):

- Inspect your code securely, without risking your system. Submit your low-level code snippets directly into SLAIT, choose between x86 or Java bytecode, and analyze them without ever putting your device into a vulnerable state.
- Track registers and flags at the moments that matter. Mark the exact points in your code where you want to observe registers or flags. SLAIT automatically records its state before, during, and after execution, so you get the insights you need.

- Run your code in a safe backend environment. Your snippets are executed inside a Docker-based sandbox. That means no matter what your code does, it stays isolated and cannot damage your personal machine.
- See clear results and compare runs easily. View an organized timeline of how your registers and flags change over execution. Compare different runs with varied initial states, spot errors you didn't expect, and measure runtime details, all presented in a way that helps you make faster decisions.

Novel features/functionalities:

- Most low-level inspection tools either require unsafe local execution or lengthy configuration of VMs/sandboxes. SLAIT provides secure, on-demand inspection in a streamlined way. The combination of Docker isolation, fine-grained register/flag tracking, and comparative visualization in a user-friendly Angular frontend makes SLAIT unique compared to existing tools, which usually do not come equipped with as much visual utility in reference to flag and register observation.

Algorithms and tools: potentially useful algorithms and software tools

- Docker
  - Sandbox to execute code within
  - Easy to create new instances
  - Low overhead in comparison to a VM
- Angular
  - Effective tool for front-end development
  - Useful for splitting independent features into their own components
- Compiler toolchains
  - GCC/JVM - executing x86 and Java bytecode
- Parsing libraries
  - Inserting observation markers in user code
- Visualization libraries
  - (D3.js, Angular charting libraries) - helpful for timeline/register visualization

Technical Challenges: Discuss three main CSE-related challenges.

- We plan to utilize Docker environments, but we are not extensively knowledgeable on Docker environments.
- We plan to work with Angular, but we have not worked with Angular before.
- We plan to utilize a backend server, but we have not had much experience with frontend backend development.
- Additional challenge: ensuring correct parsing/instrumentation of user code for register/flag observation.

Milestone 1 (Sep 29): itemized tasks:

- Compare and select technical tools for:
    - Frontend (Angular components, visualization libraries)
    - Backend (Docker containerization setup, server framework)
    - Code parsing/compilation (supporting x86 and Java bytecode)
- Provide “hello world” demos for basic frontend and backend
    - Running x86 code inside a Docker environment
    - Angular frontend component displaying sample register/flag data
    - Basic backend–frontend communication test
- Investigate technical challenges:
    - Initial Docker configuration and sandbox permissions
    - Ensuring flagged sections of code are properly translated after compilation
    - Basic Angular build/test pipeline
    - Setup of backend and backend compilation for one language (x86 assembly)
- Create Requirement Document
- Create Design Document
- Create Test Plan

Milestone 2 (Oct 27): itemized tasks:

- Implement, test, and demo:
    - User input of code snippets (frontend → backend)
    - Docker backend execution of simple x86 code with register states
    - Angular visualization of registers/flags
    - Error handling for compilation/runtime issues
- Add basic Java bytecode support (demo run)
    - Ensure Frontend parsing between different languages
- Improve Docker security (restrict system calls)
- Update Requirement and Design Documents

Milestone 3 (Nov 24): itemized tasks:

- Implement, test, and demo:
    - Marking registers/flags at code locations and comparing state changes
    - Timeline view of register/flag states
    - Comparative run mode (different initialization states)
    - Enhanced error reporting and flag tracking
- Polish Angular frontend visualization widgets
- Implement frontend comparison of multiple executions using visual graphs
- Continue frontend–backend integration (end-to-end flow)
- Prepare semester 1 progress presentation and documentation

Task Matrix for Milestone 1:

Task	Michael	Maria
Compare and select technical tools	Frontend (Angular, visualization libraries)	Backend (Docker, server framework, compiler toolchains)
“Hello World” demos	Angular component showing sample register/flag data, assist with front to back communication	Run code inside Docker, return minimal output, assist w/ Angular
Resolve technical challenges	Set up Angular build/test pipeline	Configure Docker sandbox & compile simple x86 test
Frontend–backend communication demo	Connect Angular frontend to backend API	Establish backend API for code execution
Requirement Document	Write 50% (frontend/UI focus)	Write 50% (backend/system focus)

Design Document	Write 50% (UI/UX design, frontend flow)	Write 50% (system architecture, backend flow)
Test Plan	Write 50% (frontend/unit tests)	Write 50% (backend/Docker tests)

Approval from Faculty Advisor:

- I have discussed with the team and approved this project plan. I will evaluate the progress and assign a grade for each of the three milestones.
- Signature: _____ Date: _____