

SLAIT – Secure Language Assembly Inspector Tool

Software Design Document

Maria Linkins-Nielsen, Michael Bratcher

CSE 4101 – 1st Semester Senior Design Project

Faculty Advisor: Dr. Marius Silaghi

TABLE OF CONTENTS

1.0....	INTRODUCTION.....	3
1.1....	Purpose.....	3
1.2....	Scope.....	3
1.3....	Overview.....	3
1.4....	Definitions and Acronyms.....	3
2.0....	SYSTEM OVERVIEW.....	4
3.0....	SYSTEM ARCHITECTURE.....	4
3.1....	Architectural Design.....	4
3.2....	Design Rationale.....	4
4.0....	DATA DESIGN.....	5
4.1....	Data Description.....	5
5.0....	HUMAN INTERFACE DESIGN.....	5
5.1....	Overview of User Interface.....	5
5.2....	Screen Images.....	6
5.3....	Screen Objects and Actions.....	6

1.0 INTRODUCTION

1.1 Purpose

This software design document describes the early conceptual architecture of SLAIT (Secure Language Assembly Inspector Tool). This document outlines the planned internal architecture, execution flow, and early user interface design concepts for running MASM code securely in a sandbox environment and returning register/flag data to the user.

1.2 Scope

SLAIT is intended to be a web application that allows users to write, submit, and analyze MASM x86 assembly code without running untrusted executables on their host systems. For this first-semester project, SLAIT focuses solely on MASM support, including:

- Inputting MASM code via a frontend
- Selecting execution inspections where register/flag states will be captured
- Securely running code inside a container
- Returning captured states and visualizing them in a clean, timeline-style interface

Future iterations may expand language support (e.g., Java bytecode) or add persistent run histories, but these are outside the scope of the current project.

1.3 Overview

This document provides a general overview of SLAIT's modular architecture, focusing on how user code flows from frontend to backend, executes securely, and returns meaningful register/flag results. It also introduces early UI design concepts for the viewer, inspection selection tool, and results visualization panel.

1.4 Definitions and Acronyms

Term	Definition
MASM	Microsoft Macro Assembler – used for compiling x86 assembly
Inspection	A line number specified by the user where registers/flags will be captured
Snapshot	A single captured state of CPU registers/flags at an inspection
Flags	Processor status register (e.g., ZF, CF, SF, OF)

2.0 SYSTEM OVERVIEW

SLAIT's main goal is to give students and instructors a secure, streamlined tool for low-level code inspection. Instead of running MASM programs directly on a potentially vulnerable machine, students submit code through SLAIT, which runs it in a sandbox, records register and flag states at chosen points, and returns results visually.

This improves the safety and consistency of CSE 3120 assignments and reduces system configuration complexity for students.

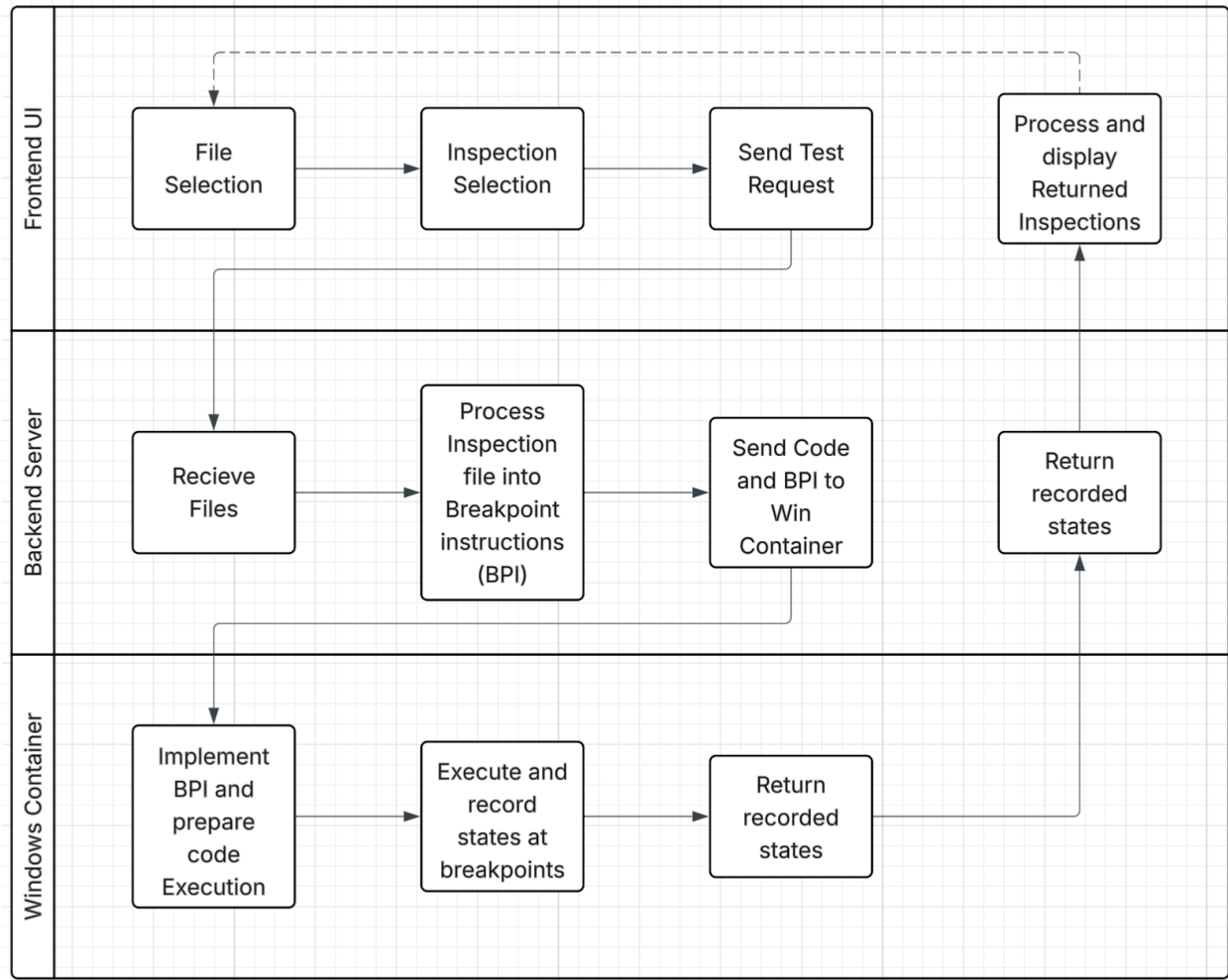
3.0 SYSTEM ARCHITECTURE

3.1 Architectural Design

SLAIT follows a simple client-server architecture with three primary layers:

1. **Frontend:** Angular-based web interface with a MASM code editor, inspection selection, and result visualization tools.
2. **Backend API:** Lightweight server that receives code, spins up a container, compiles/runs the MASM code, and gathers results.
3. **Container Execution Layer:** Prebuilt container image with MASM toolchain and debugging hooks, isolated from the host system.

Architecture Diagram:



3.2 Design Rationale

This architecture was chosen to:

- **Maximize security:** Code runs in a disposable, isolated environment.
- **Minimize setup overhead:** Students do not need to configure MASM or debuggers locally.
- **Ensure reproducibility:** Every run uses the same container image for consistent behavior.

4.0 DATA DESIGN

4.1 Data Description

The system processes three main data objects:

- **Code Input:** String containing user-submitted MASM code
- **Inspection List:** List of line numbers and booleans for registers and flags
- **Execution Results:** List of variables of each requested register/flag at each inspection

No persistent database is used in Phase 1, all data is processed in-memory and discarded after the run completes.

5.0 HUMAN INTERFACE DESIGN

5.1 Overview of User Interface

The UI will allow users to:

- Select their MASM files and view them once selected
- Add “Inspections”, where users can select lines to inspect registers and flags
- Submit code for execution
- View a results panel displaying register/flag values for each inspection
- See errors clearly highlighted in case of compilation/runtime failure

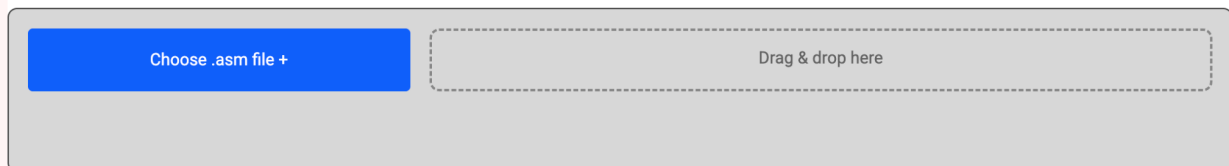
This core loop of submit code to inspection to execute to analyze results is the primary interaction flow SLAIT is built around.

5.2 Screen Images

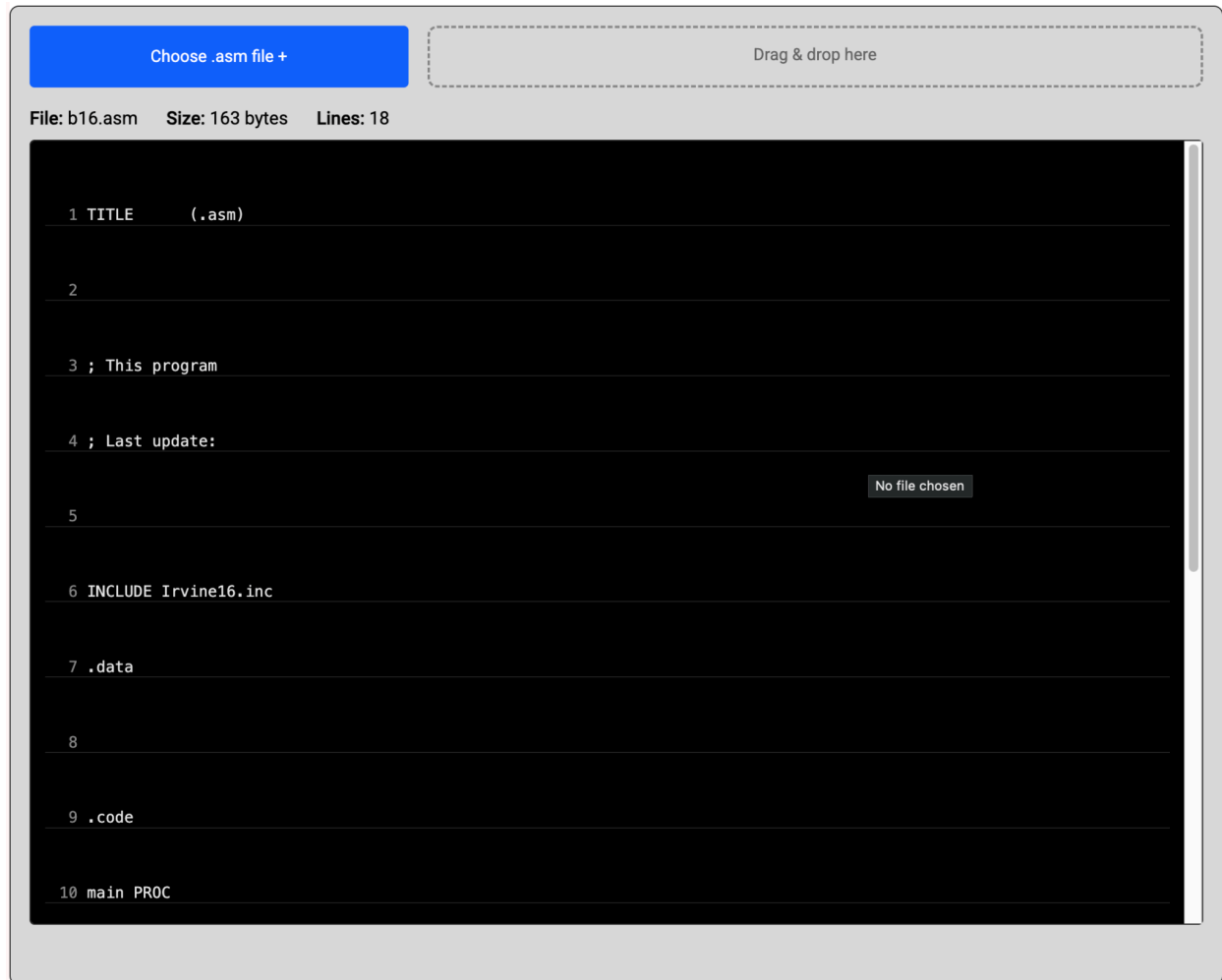
- *Code Viewer before and after file selection*

Welcome to SLAIT

Created by Michael Bratcher and Maria Linkins-Niel



The screenshot shows a user interface for file selection. It features a blue button labeled "Choose .asm file +" and a dashed rectangular box labeled "Drag & drop here". The entire interface is set against a light gray background within a rounded rectangle.



- *Inspection creator, Inspections, and Test button*

Inspections for b16.asm

Total lines: 18

Add Inspection +

Line 3 | Registers: Yes | Flags: Yes

X

Line 4 | Registers: Yes | Flags: No

X

Line 5 | Registers: No | Flags: Yes

X

Line #

☐ Registers

☐ Flags

Done

X

Test

5.3 Screen Objects and Actions

- **Code Viewer:** File viewer that only allows MASM files to be submitted
- **Inspection Creation Button:** Creates a new inspection
- **Inspection Creator:** Requests user provide line number and register/FLAG inspections
- **Created Inspections:** All inspections show user provided information in order of creation age, with an option to remove an inspection from the list
- **Test Button:** Sends code and inspection data to backend, disables while execution is in progress.
- **Results Table:** Displays snapshots of register/flag values per inspection in execution order.
- **Error Panel:** Displays structured errors if occurred.