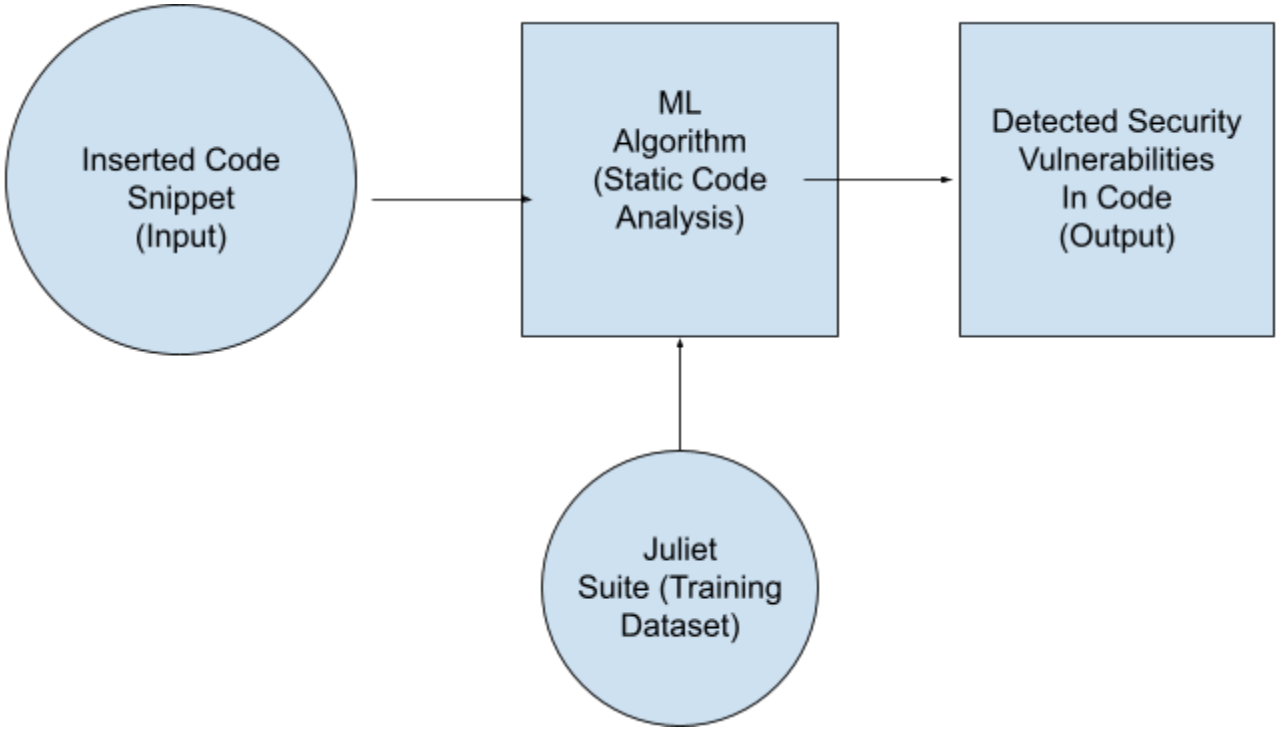
Milestone 2 Deliverable 2

Team: CEHJ - The Vulwitch Trials

Date: 10/14/2025

Application Flow:



How to Run our Application:

Our software runs in an executable file which you enter the location of your code file, the program will detect common code vulnerabilities in the code snippet provided.

Please download the executable file below to begin using our application.

Code: https://github.com/Firecon123/CSE-518-The_Vulwitch_Trials/tree/dev

Steps to download application:

1.) clone the repository from the link provided
2.) Run the command on a windows machine “pip install pysinstaller”, “pip install transformers”, “pip install torch”, “pip install keyboard”, “pip install warning”, “pip install datasets”, “pip install sklearns”
3.) Run “python -m PyInstaller main.py”

1.) After downloading the application, double click on it to open
2.) You will be prompted to enter the location of your code file, make sure this file is viable C or C++ code or the application will give you an error for entering an invalid file/location.
3.) Our program will then analyze the imported code, note this process may take a few minutes.
4.) The program will return the status of the code whether it is vulnerable or non-vulnerable, if vulnerable it will note the type of security vulnerability that needs to be fixed.

Previous Requirements:

The scope of this project was much more difficult than expected,
We were unable to get the AST trees working on the following inputs:

1. Inline assembly statements.

2. Type aliasing using `typedef` keyword.

3. Type qualifiers exclusive to MSVC, including but not limited to `__cdecl`, `__clrcall`, `__stdcall`, `__fastcall`, `__thiscall`, `__vectorcall`, `__declspec`, `__based`.

4. C11 generic selection expressions.

5. C11 static assertion.

For the ML algorithm:

1. No feedback based learning was feasible at this time
2. Limited to finding vulnerabilities in C & C++ Code

We underestimated the requirements of this project, therefore however in the break/fix stages we hope to fine tune the model and add more features that relate to our original goals set forth in 2.1.

Our AST implementation was not able to be used for our machine learning algorithm at this point, the models that we used for pattern detection took vulnerability samples at full length, not in AST format to make its predictions and train. The way to get the ML model to detect logical patterns in AST's was too abstract and difficult to implement in the 3

week time frame given. Much of the time needed was used to train the model on large datasets on the Juliet Test Suite.

We also attempted to implement a user feedback system for analysis which proved to be difficult to do the realtime analysis updates to our data. Overall though we were able to get performance metrics for our AI model as well as some form of AST's representation of the user imputed code file. Many basic C & C++ security vulnerabilities such as buffer overflow can be detected with relatively high accuracy.