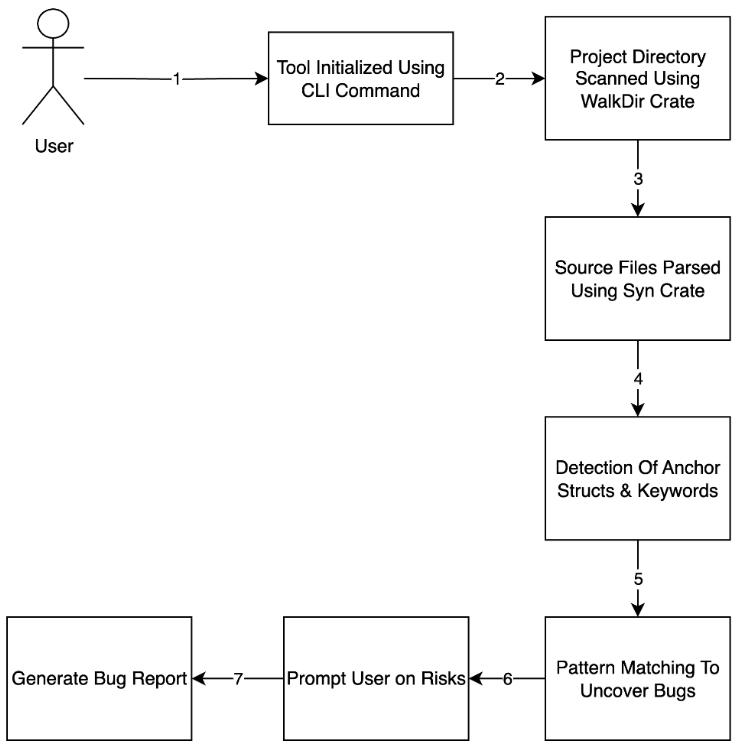
ASSIGNMENT 3: ARCHITECTURE DESIGN - JIAN YANG

Protocol POC Requirements

- The tool shall be executable via a CLI command, allowing the user to analyze an Anchor program directory
- The tool shall recursively scan the project files using **walkdir** and parse them using `syn` to extract account contexts, instruction handlers, and account definitions.
- The tool shall identify Anchor-specific structures such as **#[derive(Accounts)]** contexts and detect usage of high-risk patterns including **UncheckedAccount**, **CpiContext** or missing authority validations.
- The tool shall maintain a predefined set of bug patterns and match them against parsed program code to detect potentially unsafe logic.
- The tool shall prompt the user upon detection of suspicious patterns, allowing the user to confirm the intended behavior or flag it for further review.
- The tool shall generate a final report summarizing scanned items, matched bug patterns, and any user-acknowledged high-risk findings.

Overview


User
1
Tool Initialized Using CLI Command
2
Project Directory Scanned Using WalkDir Crate
3
Source Files Parsed Using Syn Crate
4
Detection Of Anchor Structs & Keywords
5
Pattern Matching To Uncover Bugs
6
Prompt User on Risks
7
Generate Bug Report

1. User → Initialize Tool
   – The user runs: cargo assumption-checker **<anchor-project-dir>**
   – The tool is initialized and configured to analyze the specified directory.

2. Initialize Tool → Scan Project Directory
   – The tool uses the **walkdir** crate to recursively traverse all files.
   – Only .rs files inside **/programs**, **/src**, or relevant folders are considered.

3. Scan Project Directory → Parse Source Files
   – The tool uses the **syn** crate to parse Rust source code into syntax trees**.**

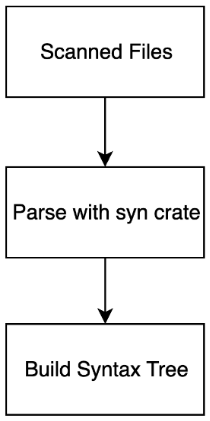
4. Parse Source Files → Detect Anchor Structures
   – Looks for **#[derive(Accounts)]** to identify account validation contexts.
   – Collects all functions under **#[program]** module for instruction handlers.
   – Maps account references within instruction handlers.

5. Detect Anchor Structures → Match Bug Patterns
   – Tool inspects the usage of critical wrappers or keywords like **UnchekedAccount**, **init**, **CpiContext::new(...)** and missing checks
   – Compares the parsed code against bug patterns

6. Match Bug Patterns → Prompt User on Risks
   – Prompts to query the user's intent with the location of the suspicious code, matched pattern name and suggestion "Is this intentional or a bug?"

7. Prompt User on Risks → Generate Report
   – A final report is generated containing total contexts scanned, user acknowledged issues, open risks

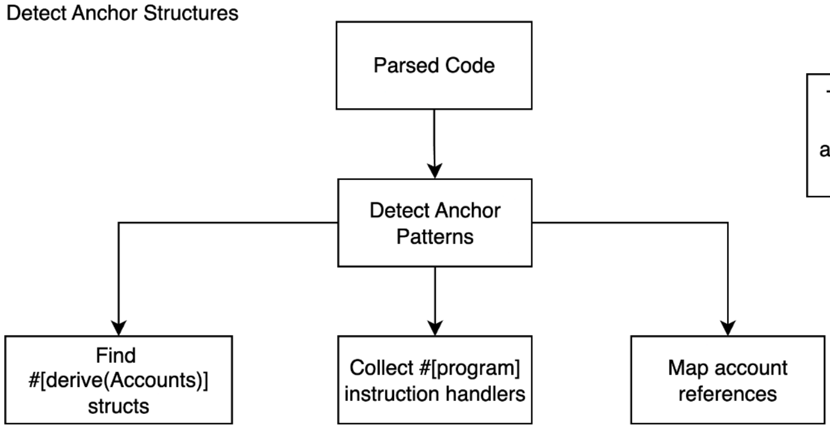
3. Parse Scanned Files



Relevant source files are then parsed with syn crate to build the Abstract Syntax Tree(AST) which makes it easier to work with the structure of the code and perform analysis on it

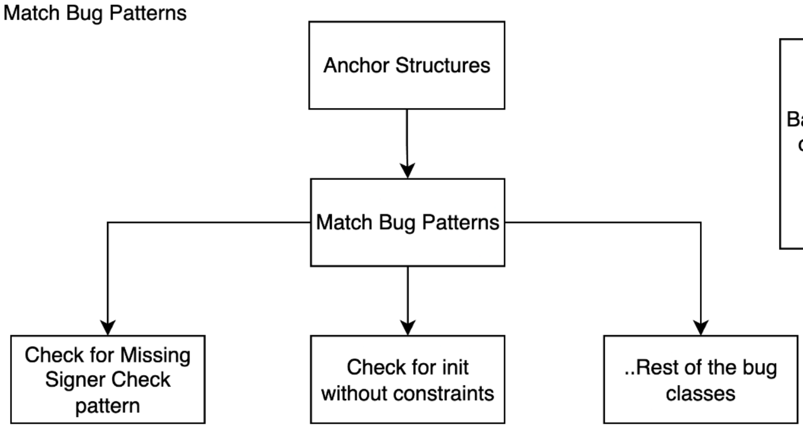
This is done using the syn crate

4. Detect Anchor Structures


Parsed Code
Detect Anchor Patterns
Find #[derive(Accounts)] structs
Collect #[program] instruction handlers
Map account references

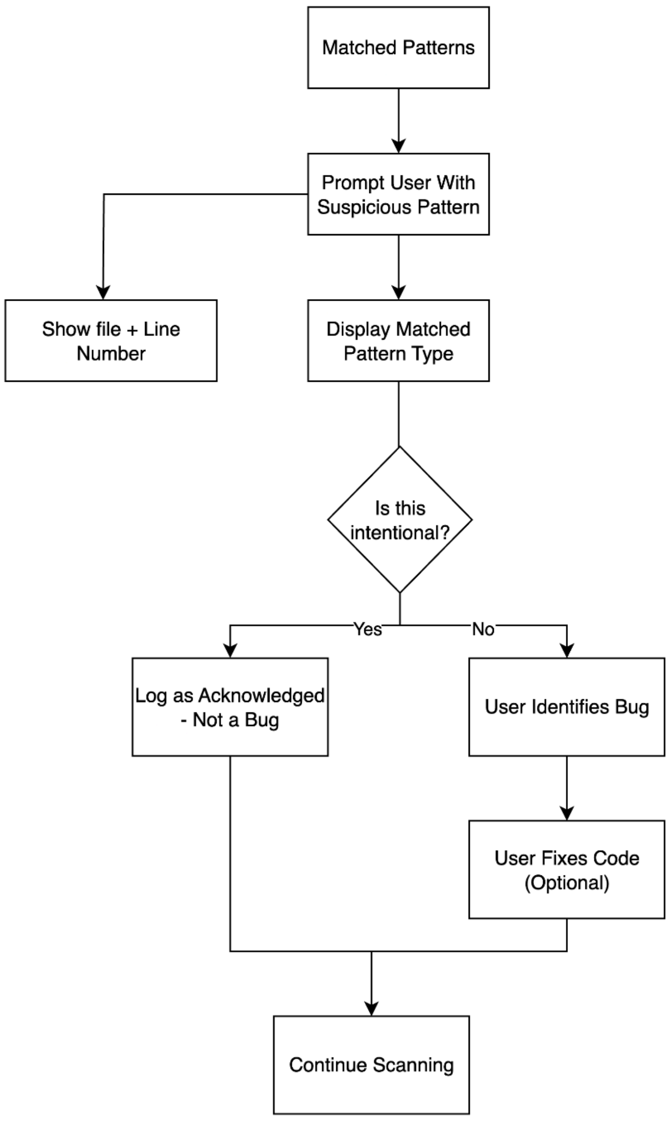
The AST is used to precisely detect patterns in the Anchor code like account validation structs, instruction handlers using #[program] etc.

5. Match Bug Patterns


Anchor Structures
Match Bug Patterns
Check for Missing Signer Check pattern
Check for init without constraints
..Rest of the bug classes

Based on the detection using AST, the code structure is matched against a set of predefined bug patterns.

6. Prompt User On Potential Risks


Matched Patterns
Prompt User With Suspicious Pattern
Show file + Line Number
Display Matched Pattern Type
Is this intentional?
Yes
No
Log as Acknowledged - Not a Bug
User Identifies Bug
User Fixes Code (Optional)
Continue Scanning

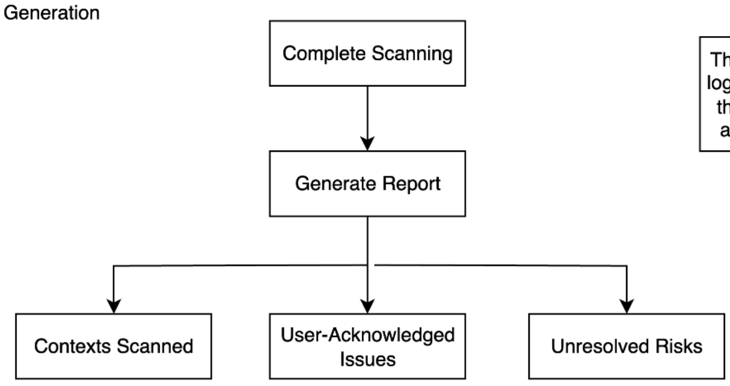
If any suspicious patterns are found, they're displayed to the User along with their location in the codebase.

User is prompted with a validation question to validate the pattern as either a bug or an intentional pattern.

In case the User acknowledges the pattern as intentional, it is logged as Acknowledged to be displayed later in the report.

In case the User identifies the flag as a bug, they can optionally fix it.

7. Report Generation



The report is generated based on the logs from the previous step along with the number of contexts, instructions and vulnerability patterns scanned.