QSE C and GSKIT Discovery Language

1. Requirement

SL#	Requirement Description
1	Scope is limited to discovery of Crypto function call in the C program
2	Parameter, data flow tracing, Vulnerability detection is completely out of scope
3	Following Libraries is in scope for now OpenSSL LibOQS CyptoPlusPlus GSKIT
4	Scope is limited to generating the below files as per the specification of R1. Metrics.json Findings.json CBOM.json DashBoard.json
5	VSCode plugin will be supported as it is for R1.

2. Architecture decisions

SL#	Decisions
1	SSA is out of scope. For initial discovery for c,c++ we don't need SSA as its typed language and scope of variable is limited to method only
2	Type resolution will not be needed as C, C++ crypto libraries functions don't have a return type for which subsequent crypto operation can be triggered. EX: Encryptor object CHAM128::Encryption encryption; encryption.SetKeyWithIV(key, sizeof(key), iv); CHAM128.Encryption exists in KB however. CHAM128.Encryption. SetKeyWithIV don't exists in DB
3	ANTLR will be used for generating the parse tree and ANTLR listeners will be used for discovery of crypto
4	Existing knowledge db entries for c and c++ will be used as it is. For GSKIT additional knowledge DB entries will be built

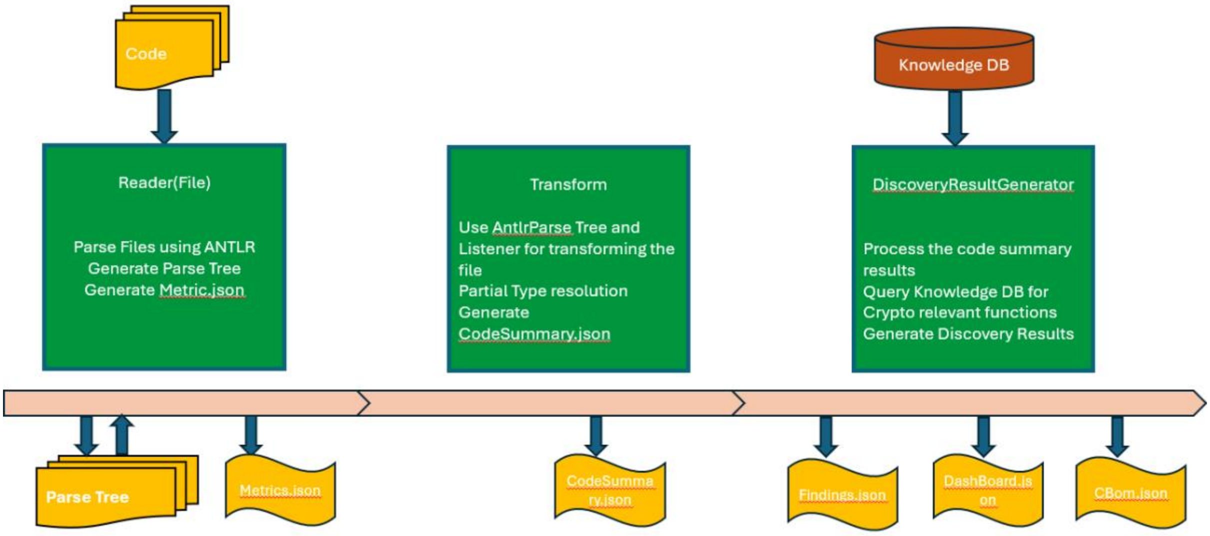
3. Performance Consideration

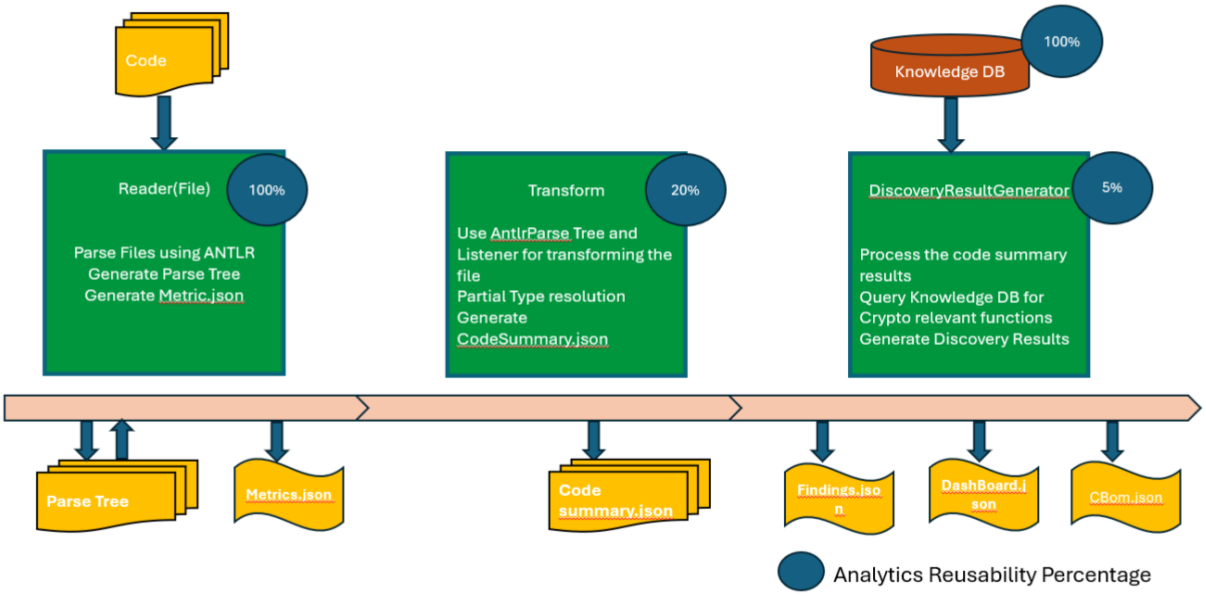
SL#	Decisions
1	Filtering of the files to be done based on the Crypto Function usage in include, using,namespace to reduce the overall numbers of files to be parsed
2	Use ANTLR listener to the extent possible rather than iterating over the parse tree to optimize the performance
3	Parallel processing of the files using ALTLR will be done to improve the performance

4. Language specific features

SL#	Features	Impact
1	In c,c++ instead of imports include, using, namespace will be used to include another C file	ANTLR listener will be used to identify include,using,namespace
2		
3		
4		
5		
6		
7		

5. High Level Architecture





6. Component Description

Name of the component	Description	Analytics Reusability Percentage
Reader (File)	It takes the files one by one. Uses ANTLR to process the file and generate parse tree. Filter/decide if the file has crypto relevant function by checking its include, using and namespaces.	80%

	Create the list of files which contains crypto relevant function and which don’t have crypto relevant functions. Pass only the files that have crypto relevant files to Transformer. Generate metrics containing total number files, total lines of code, total function etc for the entire project we are scanning. Generate the metrics.json and write it in the file system	
Transform	Process the ANTLR parse tree of only crypto relevant files. Apply different listeners to find out the variable its assignment with a function call . Create an intermediate json containing all the function with line number etc Details of the logic is described below. More accurate details to be updated later	20%
GenerateResults	Using the code summary – get the list of function calls and check in the knowledge DB if they exists. If exists, use that function call details to generate Findings.json and DashBoard.json Generate CBOM from the finding.json.	5%
Knowledge DB	Build Knowledge DB entries for GSKit Libraries	100%

7. Database Updates

DS API table and library entries for existing C++ libraries OQS, CryptoPlusPlus and OpenSSL

We will create a knowledge DB entry for GSKIT

8. Detailed Business Logic for Transformer

Logic - Step 1
To be done

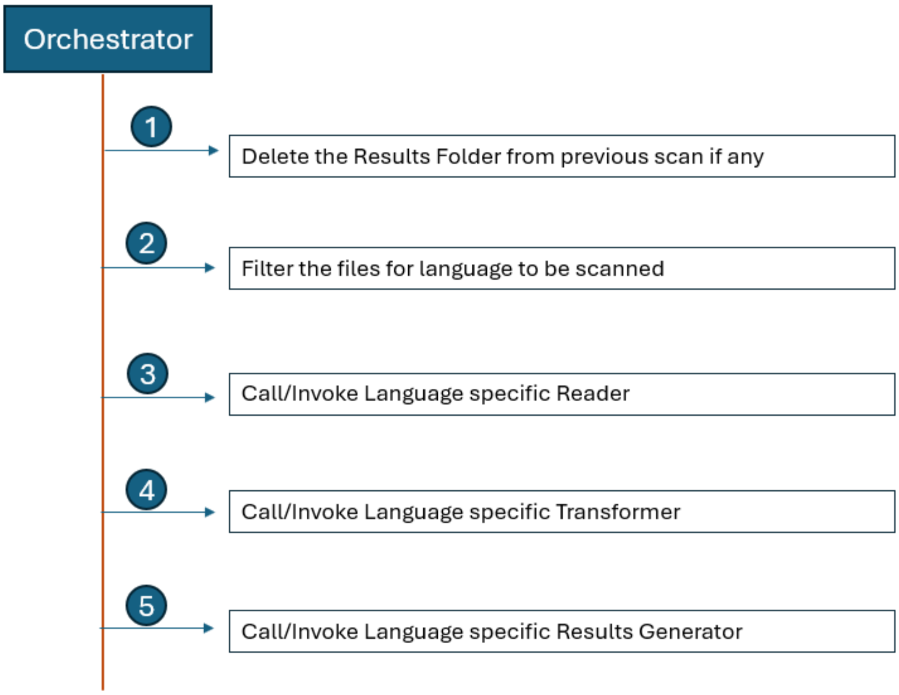
Intermediate Code Summary json Structure

```
{
 "ClassName": "Sample.c",
 "Methods": [
  {
   "MethodName": "main",
   "Variables": [],
   "Functions": [
    {
     "returnType": "Unresolved",
     "FullyQualifiedName": " CryptoPP.CHAM128Encryption ",
     "payload": " CHAM128::Encryption ",
     "line_number": 12,
     "start": 31,
     "end": 40
    },
    {
     "returnType": "Unresolved",
     "FullyQualifiedName": unresolved ",
     "payload": " encryption.SetKeyWithIV ",
     "line_number": 13,
     "start": 41,
     "end": 50
    }
   ]
  }
 ]
}
```

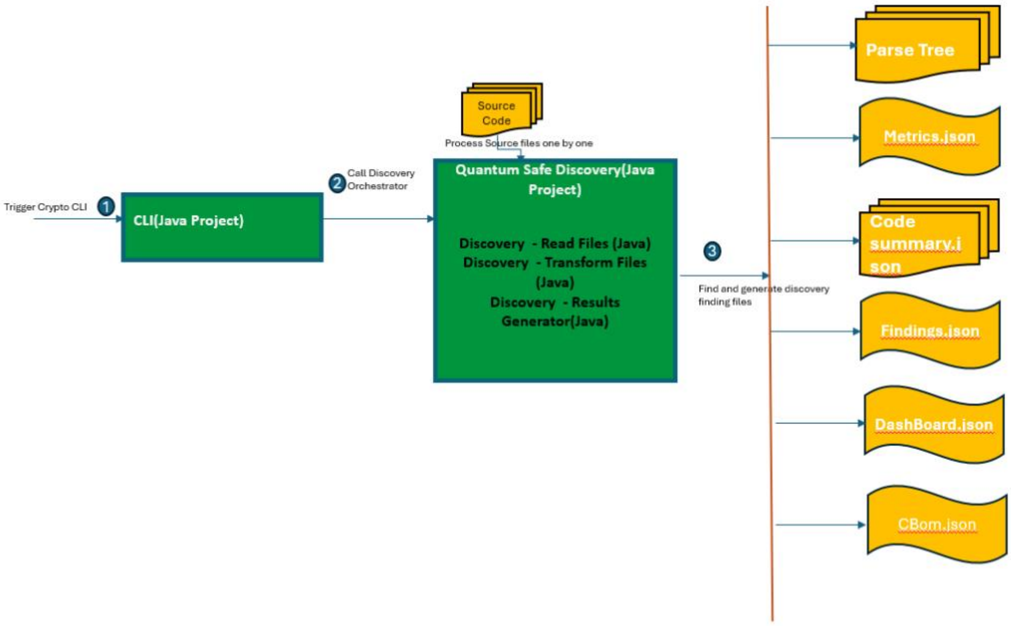
9. New File processing Orchestration – moving away from SCA.

The main objective here is to move away from OLD SCA Orchestration framework (Quantum-Safe-sca-tng project) and build the orchestration and discovery logic from scratch.

Here we will build all discovery related classes in quantum safe discovery project


Orchestrator
1
Delete the Results Folder from previous scan if any
2
Filter the files for language to be scanned
3
Call/Invoke Language specific Reader
4
Call/Invoke Language specific Transformer
5
Call/Invoke Language specific Results Generator

Projects involved in the Orchestration.


Trigger Crypto CLI
1
CLI(Java Project)
2
Call Discovery Orchestrator
Source Code
Process Source files one by one
Quantum Safe Discovery(Java Project)
Discovery - Read Files (Java)
Discovery - Transform Files (Java)
Discovery - Results Generator(Java)
3
Find and generate discovery finding files
Parse Tree
Metrics.json
Code summary.json
Findings.json
DashBoard.json
CBom.json

10. Different language Construct and Listeners

Please refer below for the list of ANTLR listeners we need to implement for crypto discovery.

SL#	Listener Names