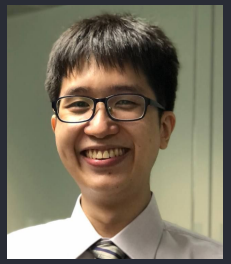
KLEE Automated Conversion Tool and Analysis

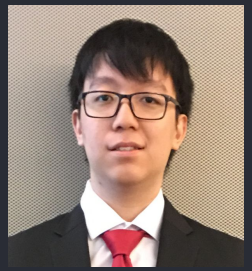
KLEENOTATION()

Cybersecurity Track
37-CS-Auto_Conversion_Tool

OUR TEAM



ALDRIC CHONG
Project Manager



ADRIAN CHANG
Lead Back-End Developer

QI XIANG
Deputy Back-End Developer

(Celine)
LOH XIAO BINN
Vulnerability Analyst

(Christine)
NGO THANH VAN
Quality Assurance

TABLE OF CONTENTS

1

ABOUT THE PROJECT

KLEENOTATION()

OUR SPONSOR

- Conducts software bug analysis through the use of a technique called symbolic execution
- Requires a command line tool (program transformation) that automates the process of launching and utilising KLEE on a software repository to aid with research
- Connect outputs from KLEE with analysis

Prof Jiang Lingxiao

Buggy or Vulnerable Programs

Testcases and output

SYMBOLIC EXECUTION

According to Carnegie Mellon University, *the process of symbolic execution is a way of executing a program abstractly, so that one abstract execution covers multiple possible inputs of the program that share a particular execution path through the code*

- Carnegie Mellon University

Gained fame since DARPA announced in 2013 the Cyber Grand Challenge

Benefits of Symbolic Execution:

- **Systematically** exploring many possible execution paths **simultaneously**
- Test whether certain properties can be violated by a piece of software

Fun Fact:
Running **24/7** in the testing process of many Microsoft applications since 2008, revealing for instance nearly 30% of all the bugs discovered by file fuzzing during the development of Windows 7

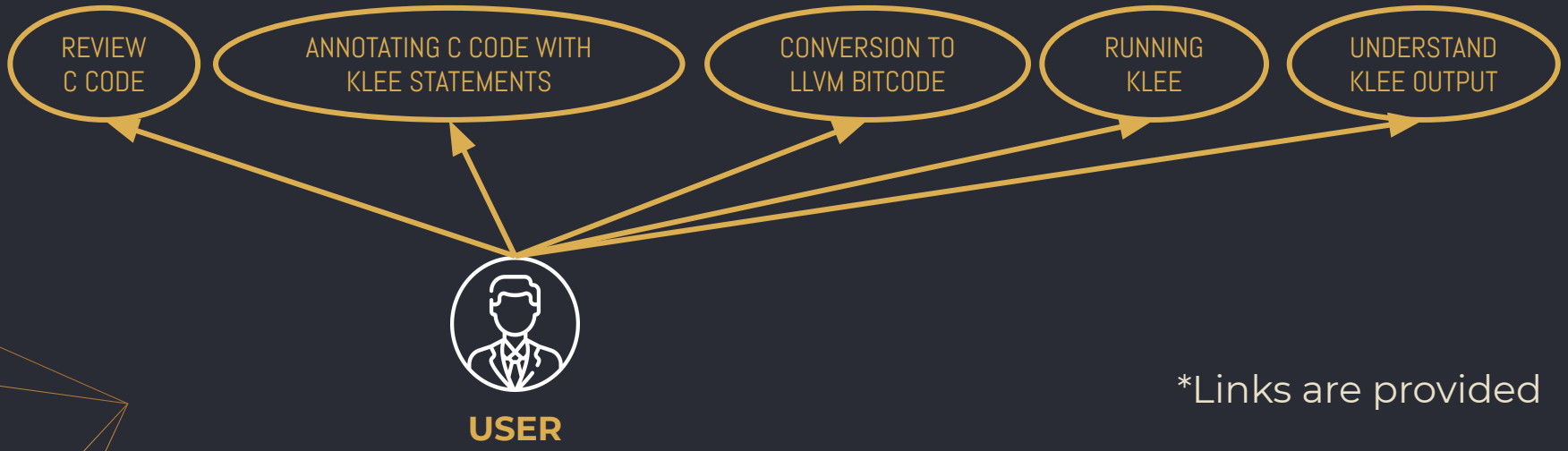
WHAT IS KLEE?

Popular dynamic symbolic execution engine, testing and analysis platform with 60 contributors on GitHub

KLEE has been used and extended by groups from many universities and companies in a variety of different areas such as high-coverage test generation, automated debugging, exploit generation, wireless sensor networks, and online gaming, among many others

CLIENT'S PAIN POINT

- Problems with current usage of KLEE
  - User has to insert relevant program lines into source code before running KLEE or using KLEE command line to input argument
  - Manual and inefficient process if working on multiple files



*Links are provided

PROJECT SCOPE

- Program Coverage: DARPA
- Primary vulnerability focus: Buffer overflow (heap & stack)
- Basic requirements
  - Preparation of source code to LLVM bitcode (.bc extension) as input to run KLEE
    - `klee file.bc`
  - Preparations mean to automate
    - Annotating the source code with klee statements
      - `klee_make_symbolic(&a, sizeof(a), "a");`
    - Annotating source code with klee dependencies
      - `#include "/home/klee/klee_src/include/klee/klee.h"`

*Links are provided

WHY KLEENOTATION()?

- KLEENOTATION() automates the instrumenting of C programs for further analysis
- KLEE: automating the use of KLEE
- NOTATION: through the use of annotating the source code
- (): open-source tool

2

REQUIREMENTS

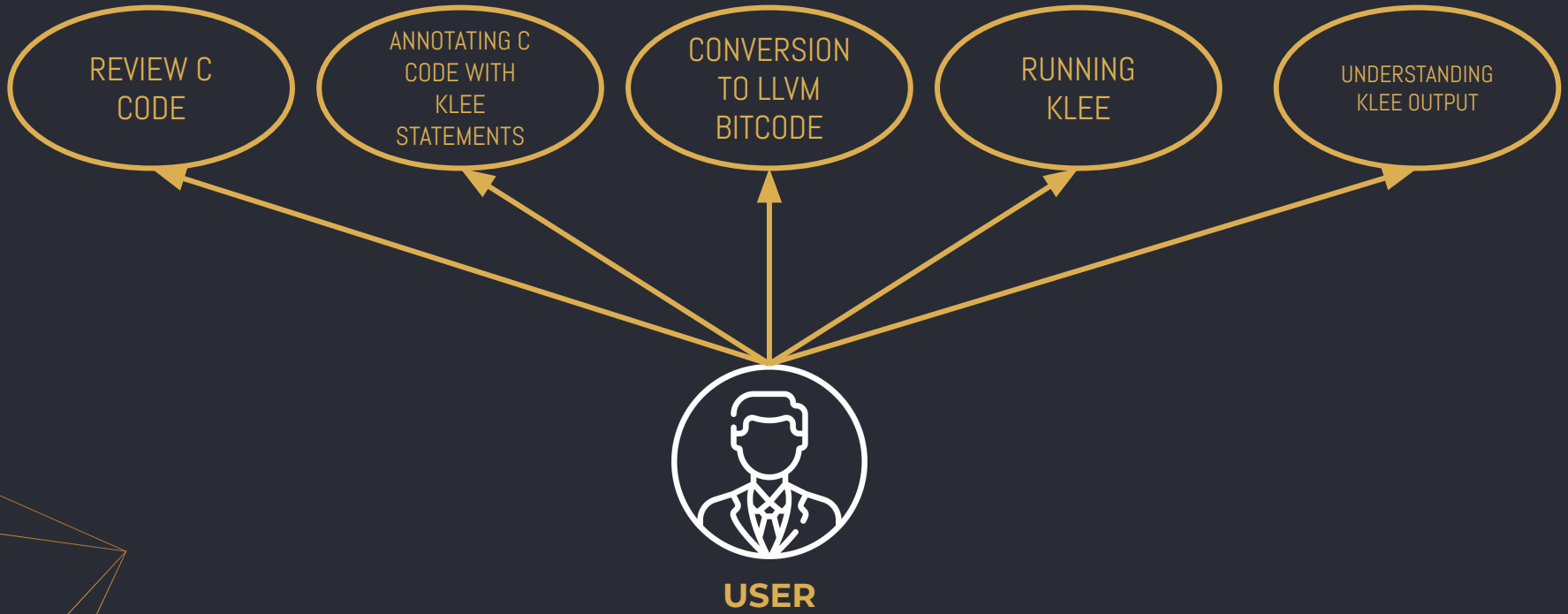
PROBLEM DESCRIPTION

- It is time-consuming and inefficient for a user of KLEE to manually:
  - Search for the variables to be made symbolic
  - Add the necessary KLEE code to make it symbolic
  - Add the necessary headers for KLEE to work
  - Run a long clang command to generate LLVM bitcode from source code
  - Run KLEE command with the necessary flags
- Our program aims to automate these steps
  - Improve efficiency
  - Get started on analysis quicker

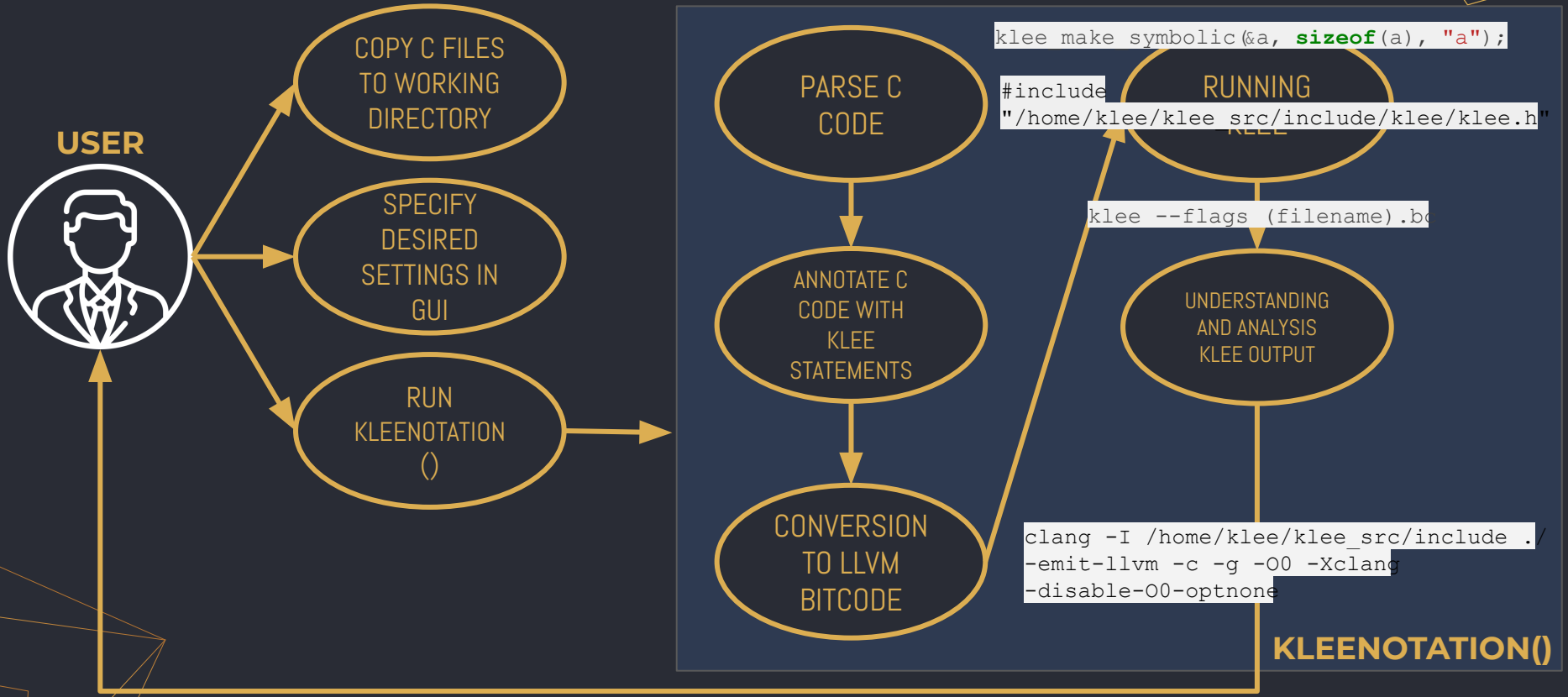
Manually

With KLEENOTATION()

USE CASE BEFORE KLEENOTATION()



USE CASE AFTER KLEENOTATION()

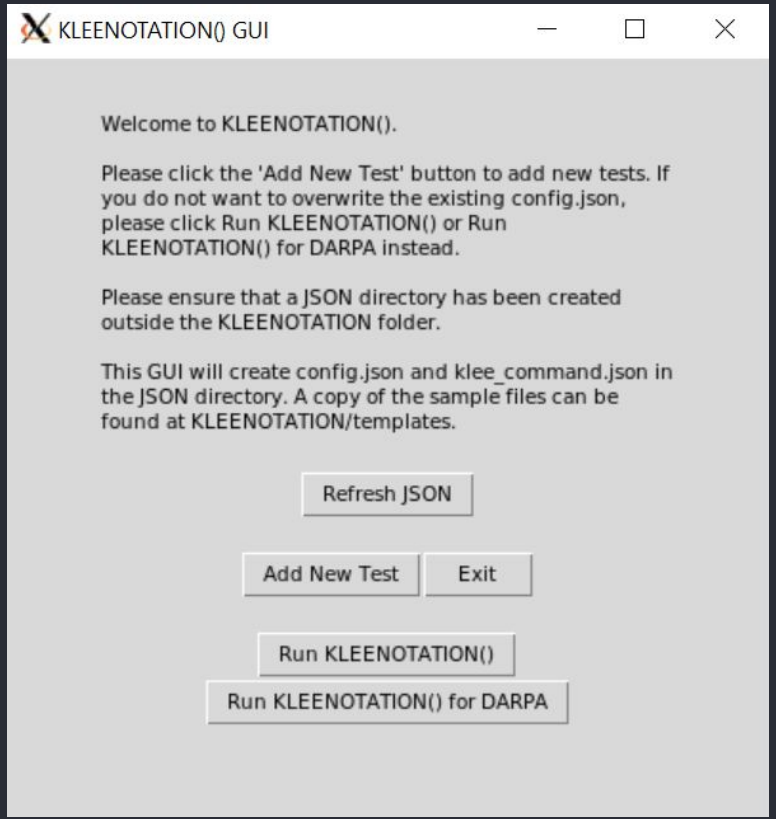


FEATURES

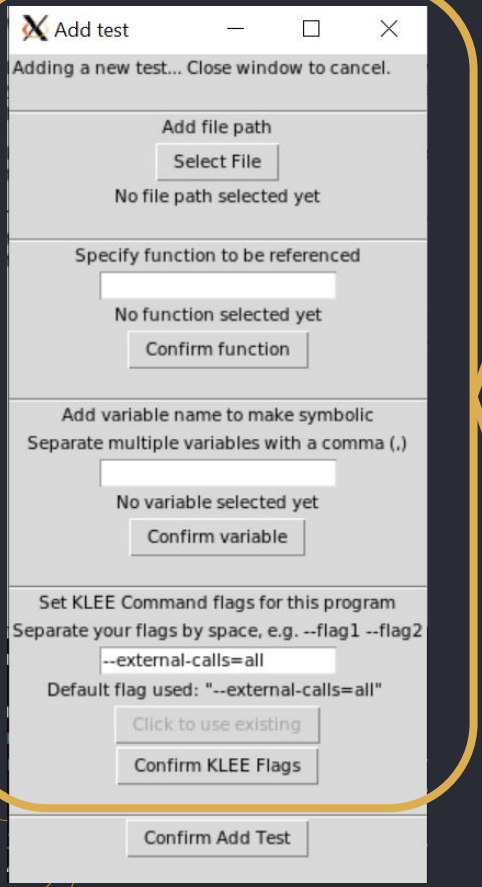
4 + 1 Modules

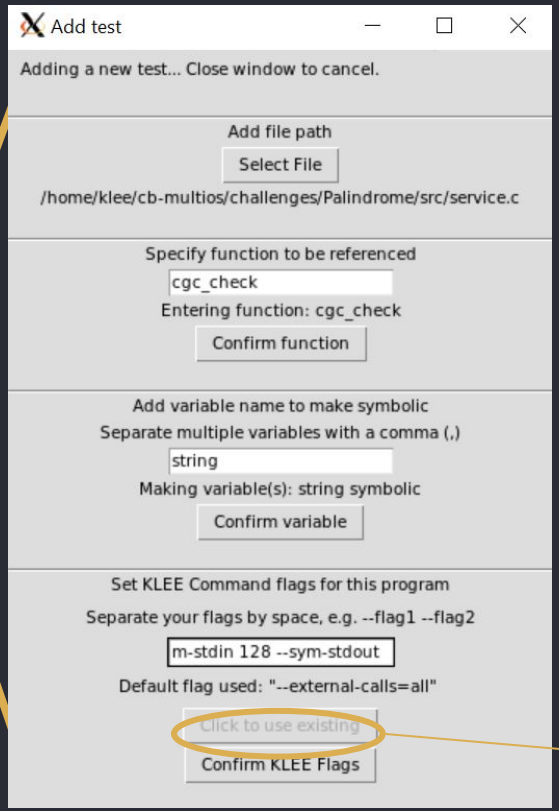
- Parser
- Annotate
- Compiler
- Analysis
- Graphical User Interface

INTERFACE - GUI



INTERFACE - GUI





Can accept multiple inputs

Will populate with previous setting when enabled

MEASURING TOOL

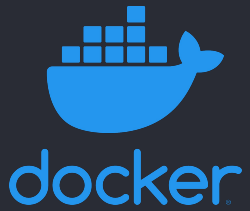
Code Type	Description	Source	Difficu Level
Simple C Code	<100 lines of code	Self-sourced	Low
Intermedia te C code	>100 lines of code	Self-sourced	Mediu
Advanced C Code (DARPA)	Built upon reference from CVE publications	DARPA CGC Corpus Challenge	High

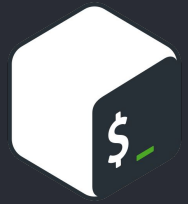
```c
#include "libcgc.h"
#include "cgc_stdarg.h"
#include "cgc_stdlib.h"
#include "cgc_string.h"
#include "cgc_ctype.h"

#define ERROR 9
#define NEW_CHALLENGE 0
#define RESET 1
#define QUIT 2

static int total = 0;
static int win = 0;

/* random.sample(open('/usr/share/dict/words', 'r').cgc_read().split('\n'), 20) */
static char* words[] = {
  "leonite", "drawdown", "conuzor", "franklin", "married",
  "excircle", "solidness", "aneuria", "constablery", "infractible",
  "speedingly", "scantlinged", "presphenoid", "diphyozooid", "twistiways",
  "didrachma", "fa", "gyte", "emblazonry", "insulize"
};

typedef struct hackman_state {
  void (*quit_handler) (void);
  void (*new_challenge_handler) (struct hackman_state *);
  char word[20];
  char progress[20];
  unsigned int num_tries;
} hackman_state_t;

int cgc_read_until(int fd, char *buf, cgc_size_t len, char delim)
{
  cgc_size_t i;
  char *c = buf;
  for (i = 0; i < len; ++i)
  {
    cgc_size_t rx;
    if (cgc_receive(fd, c, 1, &rx) != 0 || rx == 0)
      break;
    if (*(c++) == delim)
      break;
  }
  *(c-1) = '\0';
  return c - buf;
}

char cgc_parse_input(char *buf)
{
  if (cgc_read_until(STDIN, buf, 128, '\n') > 0)
  {
    switch (buf[0])
    {
```

3

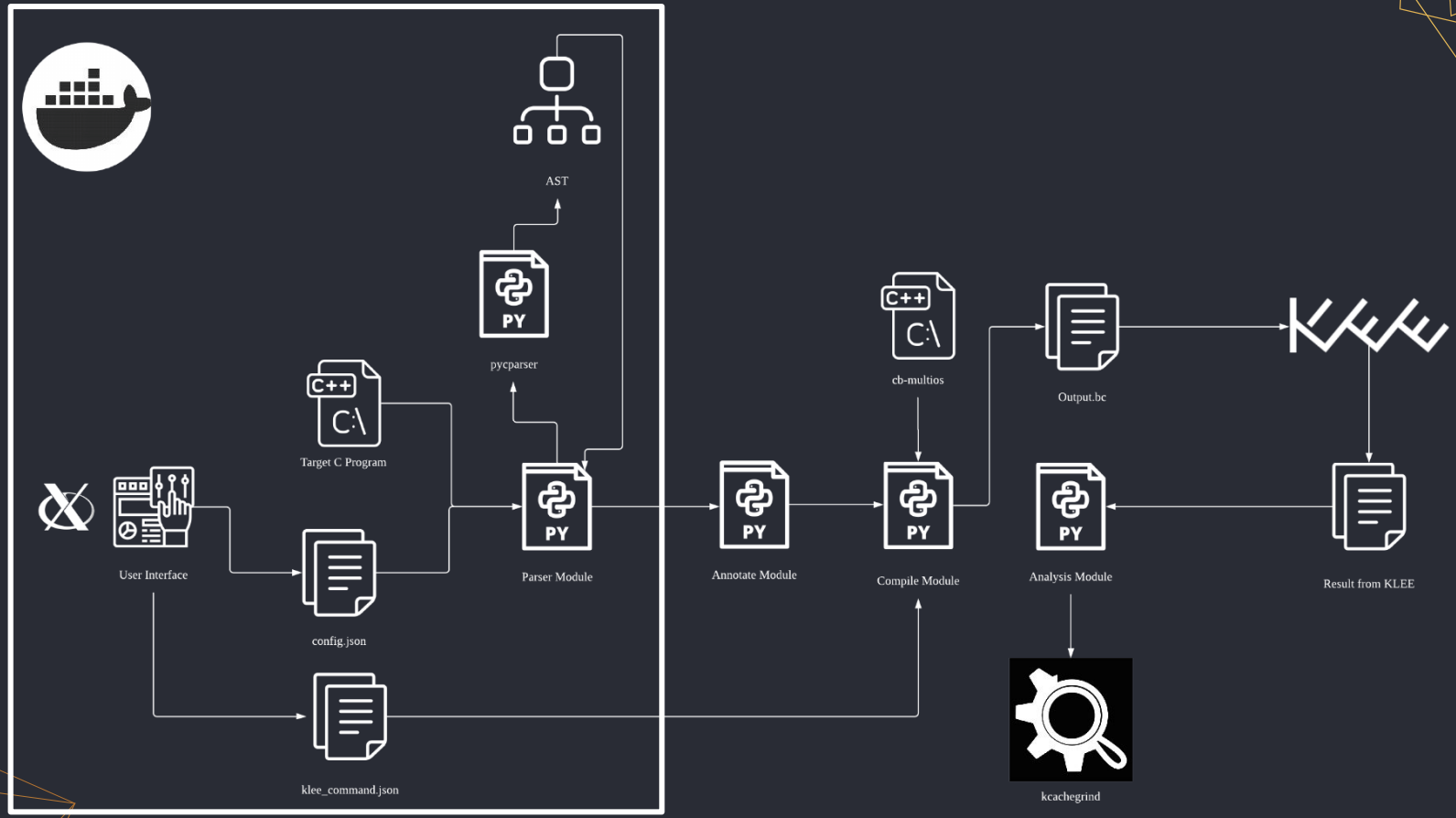
TECHNICAL DETAILS

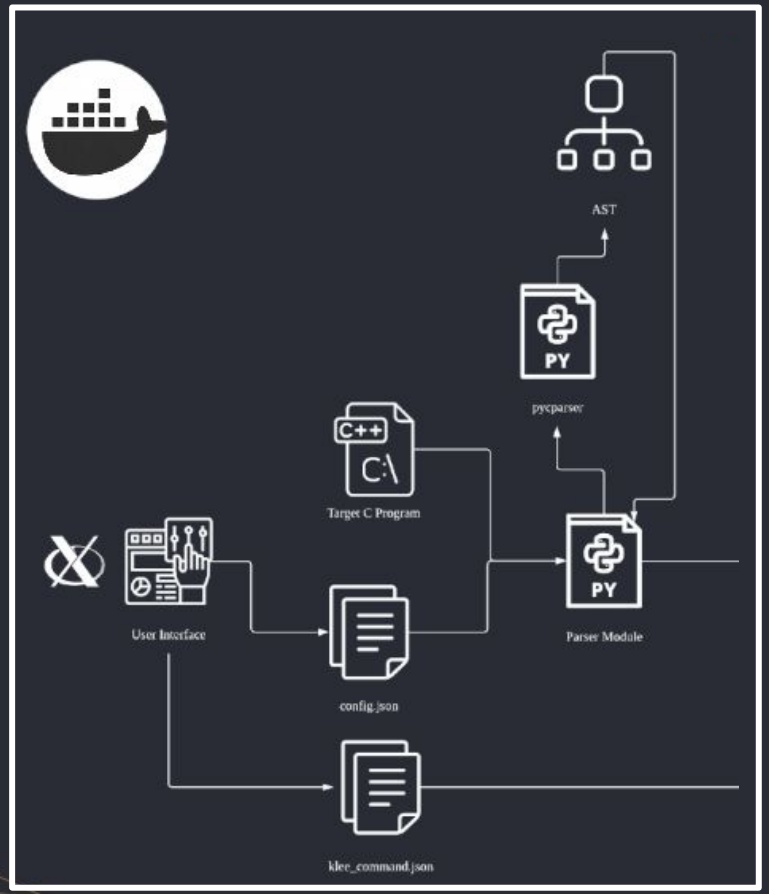
TECHNOLOGIES





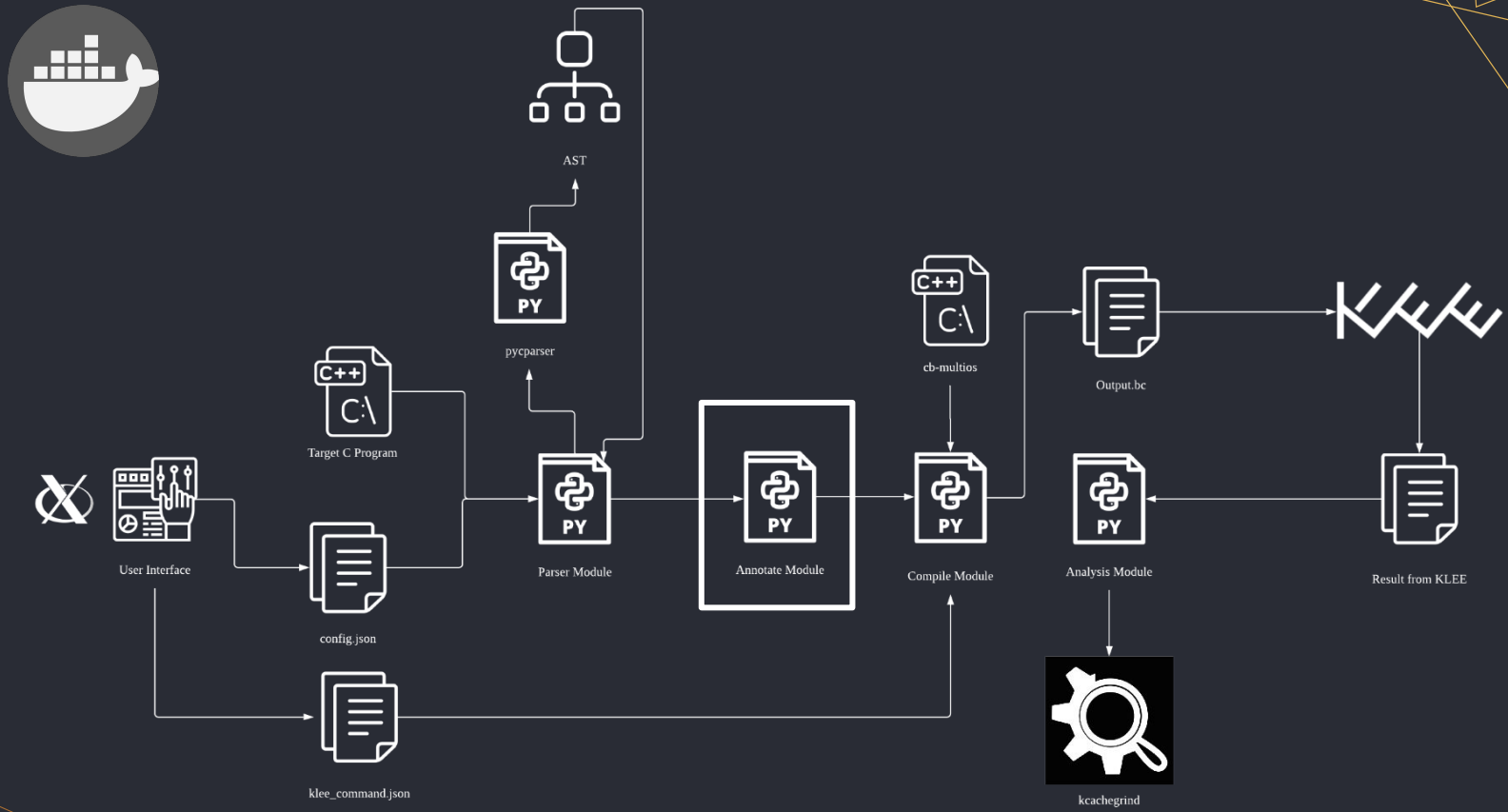
ARCHITECTURE DIAGRAM

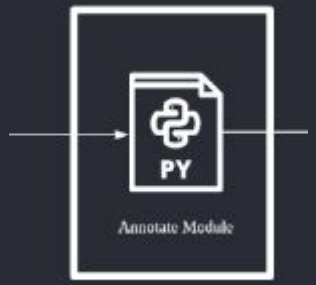




- It is time-consuming and inefficient for a user of KLEE to manually:
  - Search for the variables to be made symbolic
  - Add the necessary KLEE code to make it symbolic
  - Add the necessary headers for KLEE to work
  - Run a long clang command to generate LLVM bitcode from source code
  - Run KLEE command with the necessary flags

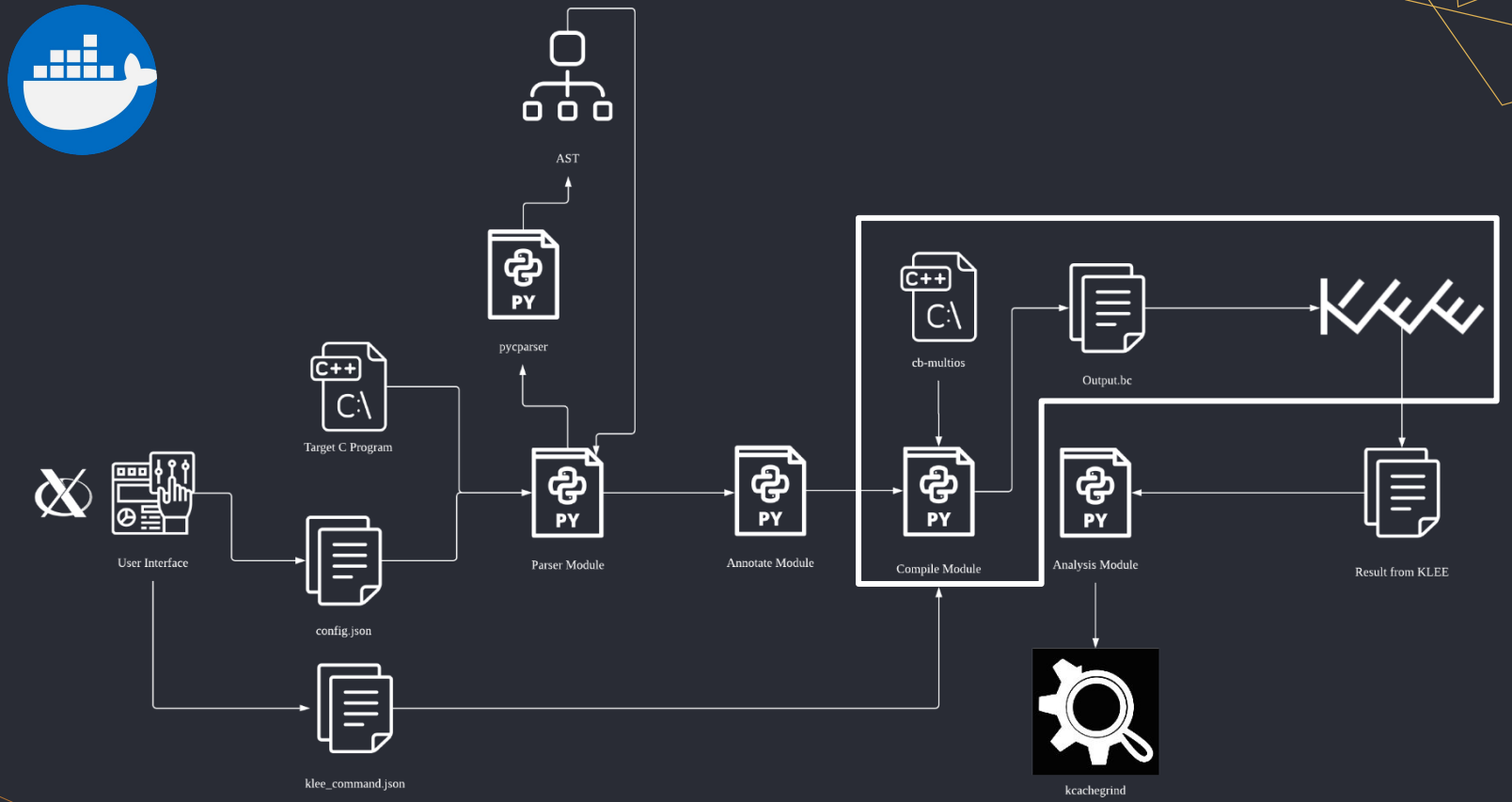
ARCHITECTURE DIAGRAM

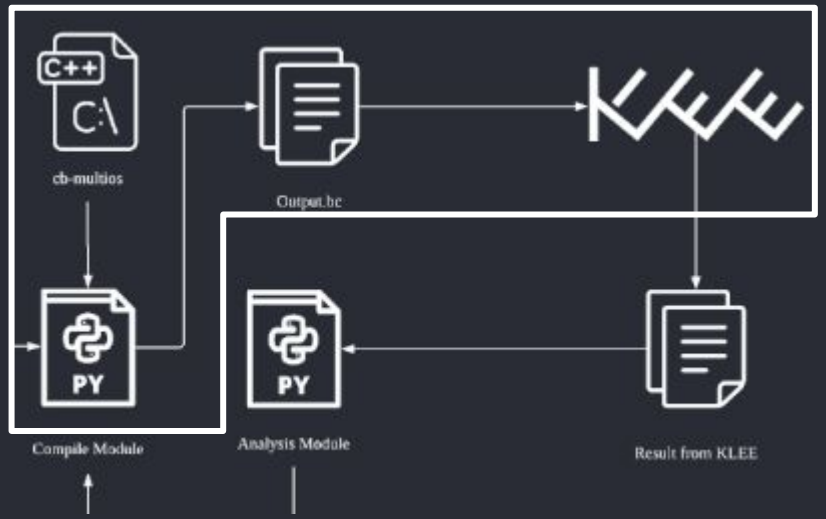




- It is time-consuming and inefficient for a user of KLEE to manually:
  - Search for the variables to be made symbolic
  - Add the necessary KLEE code to make it symbolic
  - Add the necessary headers for KLEE to work
  - Run a long clang command to generate LLVM bitcode from source code
  - Run KLEE command with the necessary flags

ARCHITECTURE DIAGRAM

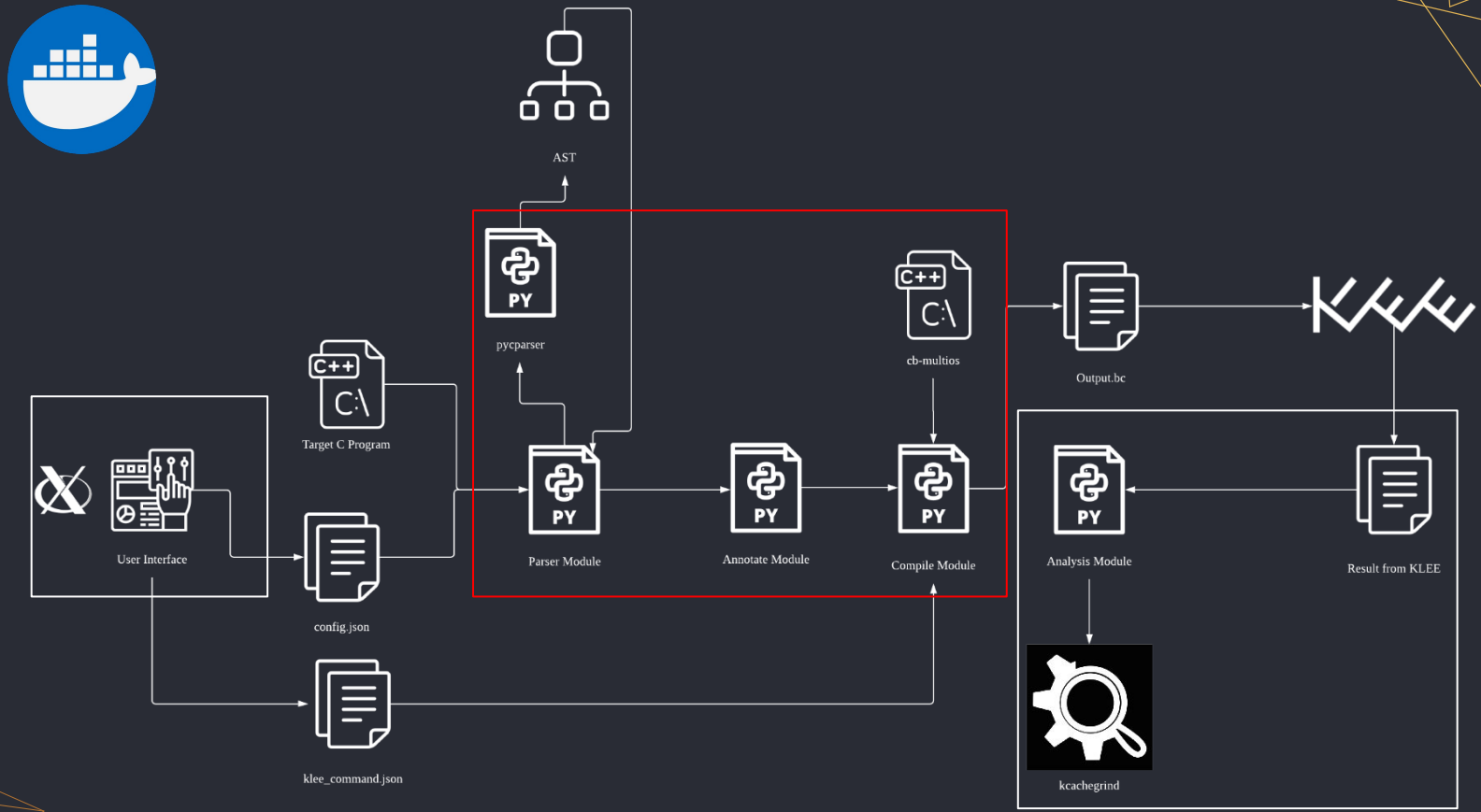




- It is time-consuming and inefficient for a user of KLEE to manually:
  - Search for the variables to be made symbolic
  - Add the necessary KLEE code to make it symbolic
  - Add the necessary headers for KLEE to work
  - Run a long clang command to generate LLVM bitcode from source code
  - Run KLEE command with the necessary flags

[ ] → New Addition

ARCHITECTURE DIAGRAM



QUALITY ATTRIBUTES (ISO 25010)

Quality model used to determine quality attributes

Product Quality - ISO/IEC 25010

Characteristics	Sub-Characteristics	Definition
Functional Suitability	Functional Completeness	degree to which the set of functions covers all the specified tasks and user objectives.
	Functional Correctness	degree to which the functions provides the correct results with the needed degree of precision.
	Functional Appropriateness	degree to which the functions facilitate the accomplishment of specified tasks and objectives.
Performance Efficiency	Time-behavior	degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.
	Resource Utilization	degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.
	Capacity	degree to which the maximum limits of the product or system, parameter meet requirements.
Compatibility	Co-existence	degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product.
	Interoperability	degree to which two or more systems, products or components can exchange information and use the information that has been exchanged.
Usability	Appropriateness recognisability	degree to which users can recognize whether a product or system is appropriate for their needs.
	Learnability	degree to which a product or system enables the user to learn how to use it with effectiveness, efficiency in emergency situations.
	Operability	degree to which a product or system is easy to operate, control and appropriate to use.
	User error protection	degree to which a product or system protects users against making errors.
	User interface aesthetics	degree to which a user interface enables pleasing and satisfying interaction for the user.
	Accessibility	degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.

QUALITY ATTRIBUTES (ISO 25010)

```
from annotate_JSON import annotate
from clang import clang
```

- **Modularity**
  - Extensible
    - New modules can easily be added to extend functionality
- **Modifiability**
  - Scalable
    - Use of JSON format for user input
      - Possibility of being scaled up to a web app in the future
      - JSON is suitable for REST API
- **Operability**
  - Easy to use UI
  - Status message output to console
    - Informs user of current process should intervention be needed

```
{
"uat_test1.c":
        [
        {"main": "s"}
        ],
"uat_test2.c":
        [
        {"main": "g"}
        ]
}
```

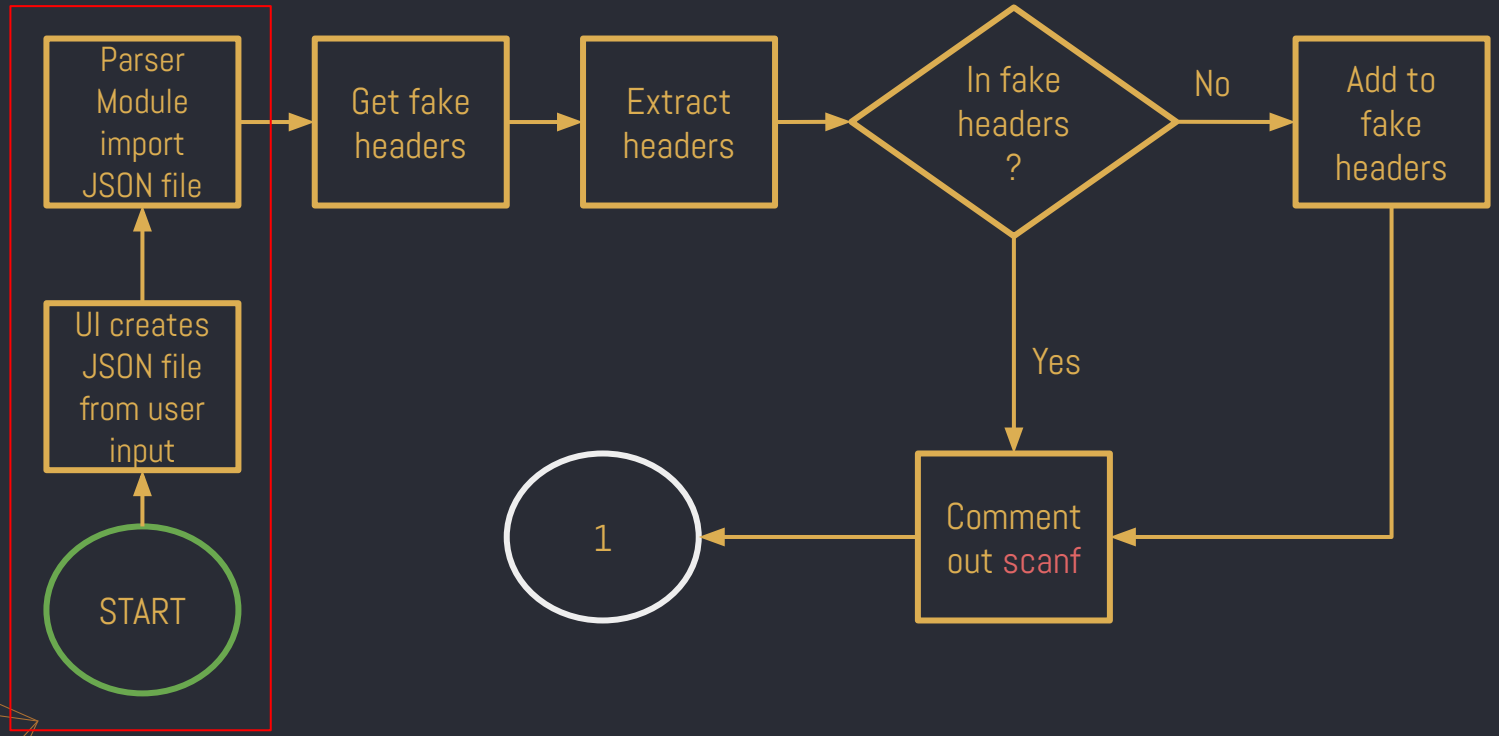
```
[*] Checking existing fake headers [*]
[*] Handling file headers of /home/klee/cb-multios/challenges/Palindrome/src/service.c [*]
 [!] missing headers located. Now generating headers for pycparser [!]
 [!] missing headers located. Now generating headers for pycparser [!]
```

QUALITY ATTRIBUTES (ISO 25010)

- **Functional Correctness**
  - Code passes all user tests to date
- **Portable**
  - Coded on Docker, can be exported

[ ] → New Addition

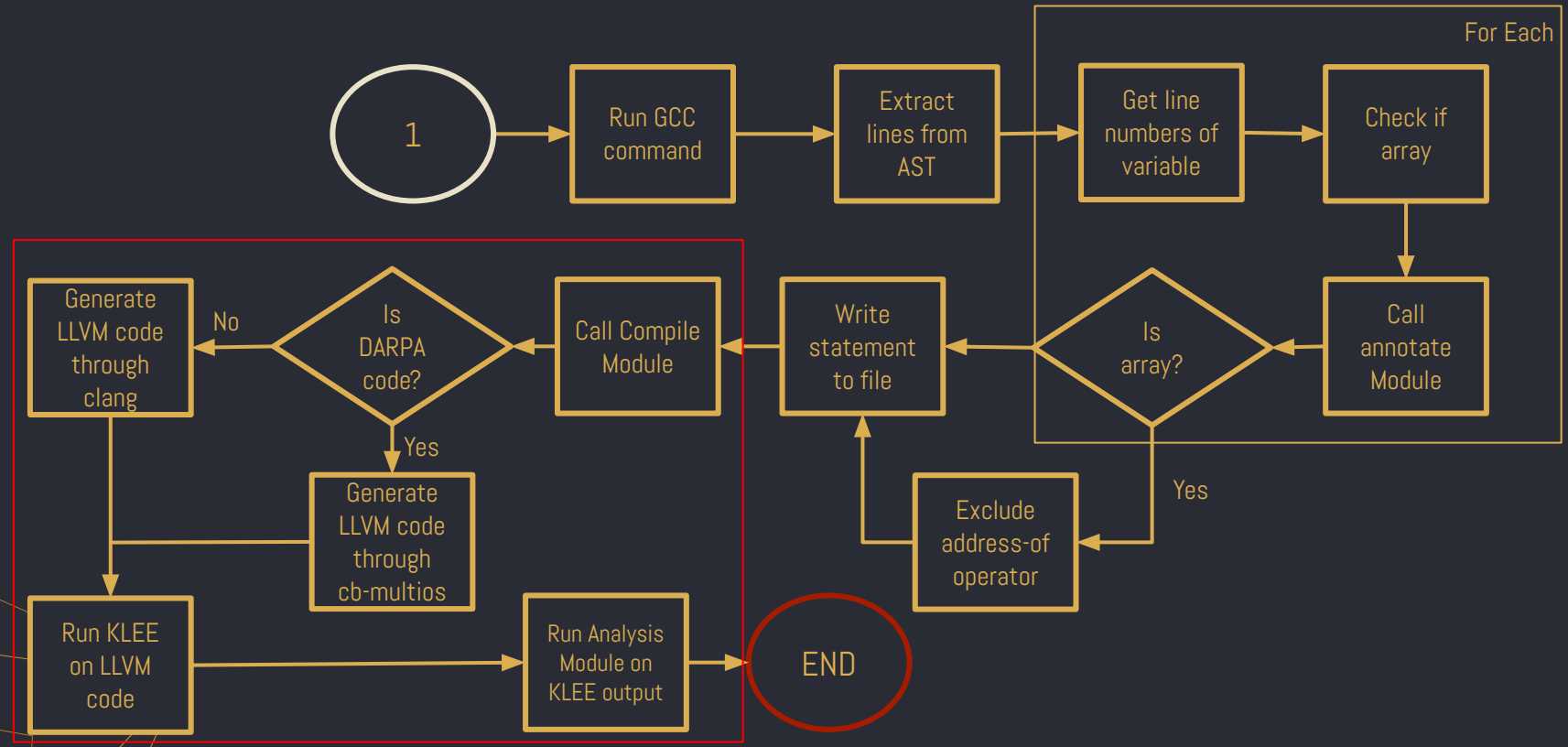
ALGORITHM



To be continued...

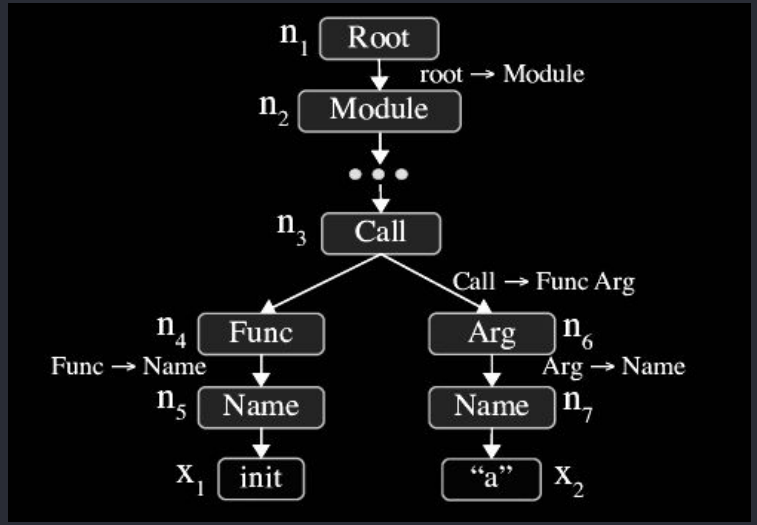
[ ] → New Addition

ALGORITHM



PARSER

Generates AST from code



PARSER

Generates AST from code

```
FuncDef:  (at test.c:3:5)
  Decl: main, [], [], [] (at test.c:3:5)
    FuncDecl:  (at test.c:3:5)
      TypeDecl: main, [] (at test.c:3:5)
        IdentifierType: ['int'] (at test.c:3:1)
  Compound:  (at test.c:3:1)
    Decl: number1, [], [], [] (at test.c:5:9)
      TypeDecl: number1, [] (at test.c:5:9)
        IdentifierType: ['int'] (at test.c:5:5)
    Decl: number2, [], [], [] (at test.c:5:18)
      TypeDecl: number2, [] (at test.c:5:18)
        IdentifierType: ['int'] (at test.c:5:5)
```

PARSER

Perform text search to obtain the line where the variable is declared

```
TypeDecl: number2, [] (at test.c:5:18)
```

PARSER

Comment out occurrences of scanf

```
## Comment out SCANF & gets ##
content = []
with open(filename, "r") as f:
    lines = f.readlines()
    for line in lines:
        if "scanf(" in line and "/*" not in line:
            content.append("/*" + line + "*/")
        if "gets(" in line and "/*" not in line:
            content.append("/*" + line + "*/")
        else:
            content.append(line)
```

ANNOTATE

Add required commands

```
data2.insert(0, '#include "/home/klee/klee_src/include/klee/klee.h"\n')
```

```
data2.insert(line_number, 'klee_make_symbolic('+ sign + variable_name +
', sizeof(' + variable_name + '), \"' +  variable_name + '\");\n')
```

COMPILE

Run required clang command, to obtain .bc file

```
clang -I/home/klee/klee_src/include -emit-llvm -c -g -O0 -Xclang
-disable-O0-optnone -I./ (filename)
```

COMPILE (DARPA)

Run make via `build.sh (in cb-multios)` → obtain binary files

Extract .bc files via `extract-all-bc-files.sh`

```
#!/usr/bin/env bash
cd build64/challenges
i=1
for folder in *
do
    echo "-"
    echo "Now processing No.$i $folder"
    cd $folder
    extract-bc $folder
    cd ..
    echo "+"
    echo ""
    let "i++"
done
```

ANALYSIS MODULE

Extract information from .err files

```
Error: memory error: out of bound pointer
File: testoob.c
Line: 13
assembly.ll line: 31
Stack:
        #031 in main () at testoob.c:13
```

ANALYSIS MODULE

Cross-reference with REFERENCE_DICT

```
REFERENCE_DICT = {
  "memory error: out of bound pointer" : ["Buffer Overflow Vulnerability", "Significant",
}
```

Scope	Impact
Availability	**Technical Impact:** *Modify Memory; DoS: Crash, Exit, or Restart; DoS: Resource Consumption (CPU); DoS: Resource Consumption (Memory)* Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
Integrity Confidentiality Availability Access Control	**Technical Impact:** *Modify Memory; Execute Unauthorized Code or Commands; Bypass Protection Mechanism* Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.
Integrity Confidentiality Availability Access Control Other	**Technical Impact:** *Modify Memory; Execute Unauthorized Code or Commands; Bypass Protection Mechanism; Other* When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Possible Impact of Buffer Overflow (From CWE, MITRE)

ANALYSIS MODULE

Provide output

```
[!] In file testoob.c, the following errors have occured:
memory error: out of bound pointer: Appears 1 times.
[!] There is a low probability that there is a Buffer Overflow Vulnerability. Impact: Significant
[i] Based on DARPA repository code analysis, out of 134 memory errors, there were 43 Buffer Overflows (Occurence rate: 32.09
%)
[i] Remediation:

Strategy: Environment Hardening (CWE-120):

 When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed in
put values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

 Strategy: Compilation or Build Hardening (CWE-121):

 Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates o
r eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mec
hanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY
_SOURCE GCC flag, and StackGuard.
```

4

TESTING

TESTING SCHEME

Four levels of testing:

- Unit testing (every change in the code)
- Integration testing (once module is complete and tested in unit testing)
- System testing (once all modules are complete and tested in integration testing)
- User acceptance testing (once system is fully integrated and tested in system testing)

TESTING SCHEME

Benefits of testing scheme:

- Ensure that all modules work properly
- Detect, log, and fix bugs and defects in the code early
- Ensure wide testing coverage
- Simulate real-world business use cases
- Identify opportunities for code/module improvements

UNIT TESTING

Unit testing continuously done for:

- Parser
- Annotate
- Compiler
- Analysis
- Graphical User Interface (GUI)

UNIT TESTING

Date	Test File name	Test case	Expected output	Actual output	Pass/Fail	Severity
6 Mar	CROMU_00001.c	File name: CROMU_00001.c Variable name: message Function name: handle_loggedin	Line number: 555 Array variables: ['message']	Output file CROMU_00001_out.c is generated Line 557 in the output file has been changed to `klee_make_symbolic(message, sizeof( message ), "message");`	Pass	
	CADET_00001.c	File name: CADET_00001.c Variable name: string Function name: main	Line number: 47 Array variables: ['string']	Output file CADET_00001_out.c is generated Line 49 in the output file has been changed to `klee_make_symbolic(string, sizeof( string ), "string");`	Pass	

12 Mar	sortArray.c	{
		"sortArray.c":
		[
		{"main": ["a", "i"]}
		]
		}
			Line number:
			8
			Array variables:
			['a']
				Output file sortArray_annotated.c is generated
				Line 10 in the output file have been added with
				`klee_make_symbolic()` function
					Pass	

	CADET_00001.c	File name: CADET_00001.c Variable name: string Function name: main	Line number: 47 Array variables: ['string']	Output file CADET_00001_out.c is generated Line 49 in the output file has been changed to `klee_make_symbolic(string, sizeof( string ), "string");`	Pass	

15 Mar	sortArray.c	{
		"sortArray.c":
		[
		{"main": ["a", "i"]}
		]
		}
			Line number:
			8
			Array variables:
			['a']
				Output file sortArray_annotated.c is generated
				Line 10 in the output file have been added with
				`klee_make_symbolic()` function
					Pass	

UNIT TESTING

Date	Test File name	Test case	Expected output	Actual output	Pass/Fail	Severity
6 Mar	CROMU_00001.c	File name: CROMU_00001.c Variable name: message Function name: handle_loggedin	Line number: 555 Array variables: ['message']	Line number: 555 Array variables: ['message']	Pass	
	CADET_00001.c	File name: CADET_00001.c Variable name: string Function name: main	Line number: 47 Array variables: ['string']	Line number: 47 Array variables: ['string']	Pass	

12 Mar	sortArray.c	{
		"sortArray.c":
		[
		{"main": ["a", "i"]}
		]
		}
			Line number:
			8
			Array variables:
			['a']
				Line number:
				8
				Array variables:
				['a']
					Pass	

	CADET_00001.c	File name: CADET_00001.c Variable name: string Function name: main	Line number: 47 Array variables: ['string']	Line number: 47 Array variables: ['string']	Pass	

15 Mar	sortArray.c	{
		"sortArray.c":
		[
		{"main": ["a", "i"]}
		]
		}
			Line number:
			8
			Array variables:
			['a']
				Line number:
				8
				Array variables:
				['a']
					Pass	

USER ACCEPTANCE TESTING - Scenario

- Mr. Tan, a KLEE user, is looking for ways to speed up the process of pre-processing C programs for KLEE input
- Mr. Tan has three C programs to run symbolic analysis on
- Mr. Tan pulls the Docker image that has been deployed on KLEENOTATION() Github
- Mr. Tan runs the KLEENOTATION() tool in the command line
    - Navigate to the executable working directory
    - Copy the C files into this working directory
    - Use either the command line or graphical user interface to run the tool
    - Verify that the tool works correctly
    - Run Analysis module to view Security Analysis

```
{
"/home/klee/POC/program4/testoob.c":
        [
        {"main":["number1"]}
        ],
"/home/klee/POC/program5/test.c":
        [
        {"":[""]}
        ],
"/home/klee/cb-multios/challenges/Palindrome/src/service.c":
        [
        {"cgc_check":["string"]
        ]
}
```

4. Press Ctrl+S to save the file. Then, press Ctrl+X to exit the nano editor
5. Enter `nano klee_command.json`
6. Enter the following details into the file. Make sure all characters are correct

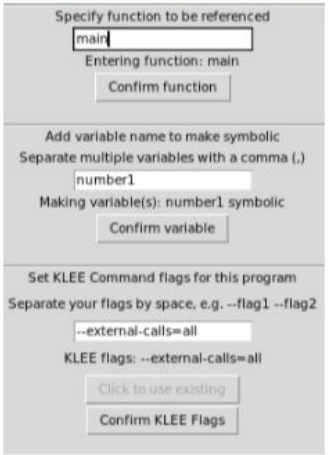
Full klee_flags command:

```
-link-llvm-lib=/home/klee/cb-multios/build64-backup/include/libcgc.so.bc
-link-llvm-lib=/home/klee/cb-multios/build64-backup/include/tiny-AES128-C
/libtiny-AES128-C.so.bc --simplify-sym-indices --write-cvcs --write-cov
--output-module --disable-inlining --optimize --use-forked-solver
--use-cex-cache --only-output-states-covering-new
--max-sym-array-size=4096 --max-time=300 --watchdog
--max-memory-inhibit=false --max-static-fork-pct=1
--max-static-solve-pct=1 --max-static-cpfork-pct=1 --switch-type=internal
--search=random-path --search=nurs:covnew --use-batching-search
--batch-instructions=10000 --posix-runtime Palindrome.bc --sym-stdin 128
--sym-stdout
```

Once inside KLEE image command line:

1. At home directory `/home/klee`, make a new directory by entering the command line `mkdir JSON`
2. At home directory, enter `cd FYP/FYP/UAT`
3. Inside ~/FYP/FYP/UAT, enter `export DISPLAY=<copied IPv4 address>:0.0`
4. Enter `python3 gui.py`

5. Click "*Add New Test*". When the second window pops up, click "*Select file*" and choose the following file at the directory **`/home/klee/POC/program4/testoob.c`**
6. Type the following inputs in the fields. Make sure all characters are correct



7. Click "*Confirm Add Test*"
8. Repeat the same steps (step 4 - step 6) for the next two test cases

USER ACCEPTANCE TESTING - Goals

- Ensure the functionality of the fully developed modules: Parser, Annotate, Compiler, Analysis, GUI works
- Ensure full integration of the modules to the system build environment
- Test the usability of the graphical user interface. Compare the performance between two interfaces

USER ACCEPTANCE TESTING - Results

Quantitative: 10 participants took part in the user acceptance testing

- Version A:
  - Average time taken: 621 seconds - 10.35 minutes
  - Average difficulty rating: 3.3/4
  - Passed tasks rate: 100%

- Version B:
  - Average time taken: 569 seconds - 9.48 minutes
  - Average difficulty rating: 2.2/4
  - Passed tasks rate: 100%

USER ACCEPTANCE TESTING - Results

Qualitative:

- 7/10 of the participants completed the tasks without or with little help from test facilitator
- Participants generally:
  - Find the graphical user interface more intuitive and easier to use than the command line interface
  - Find the tool helpful for automating pre-KLEE tasks
  - Find inputting the file path, file name, and variable name tedious and prone to mistakes when typing in the JSON file in version A
  - Like to see the complete JSON file generated by the GUI

USER ACCEPTANCE TESTING - Results

Feedback:

- Command-line interface maybe better for IT people, but GUI is great for both IT and non-IT
- Some parts are still being displayed in the command line, which could be confusing to switch
- Tasks were sped up significantly with the tool compared to the manual process

CHALLENGES & SOLUTIONS

Challenge	Solution
Limited knowledge of technology (C language, KLEE, Python libraries, etc)	• Consult project sponsor • Research and self-study • Trials and errors
Technology complexity (module integration, environment integration, environment set up, etc)	• Implement pair-coding • Continuous unit testing • Trials and errors
Limited pool of users for testings	• Talk with more people to identify prospects • Ask project sponsor for connections

5

PROJECT MANAGEMENT

CRYSTAL METHOD

The seven properties are:

1. Frequent delivery
2. Reflective improvement
3. Osmotic communication
4. Personal safety
5. Focus
6. Easy access to expert users
7. Technical environment

FREQUENT DELIVERY

Regular iteration of the programme with regular releases

- Regular settings with our sponsors
- Ensure regular feedbacks are received and actioned upon

REFLECTIVE IMPROVEMENT

- Always check back on process that is not working
- Reflects on feedback gathered from sponsors

OSMOTIC COMMUNICATION

- Development is done in small teams
- Weekly team meetings is conducted
  - Ensure that no gap in communication within the team
  - Greatly improving efficiency
  - Reduces error that arises

PERSONAL SAFETY

Safe space for communication

Every member can freely express their views

FOCUS

- Every meeting starts with updates and briefings by the whole team
- Project manager constantly checks in on teams progress
- Meetings and codings sessions are grouped into 2-hours sessions

EASY ACCESS TO EXPERT USERS

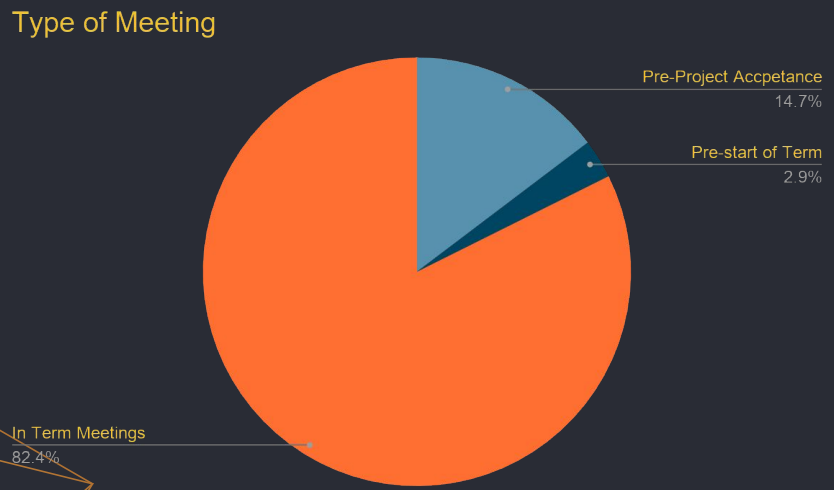
- Actual users for KLEE
  - Professor Jiang Lingxiao
  - PHD Student, Mr Tu Haoxin
- Provided helpful feedback and suggestions

TECHNICAL ENVIRONMENT

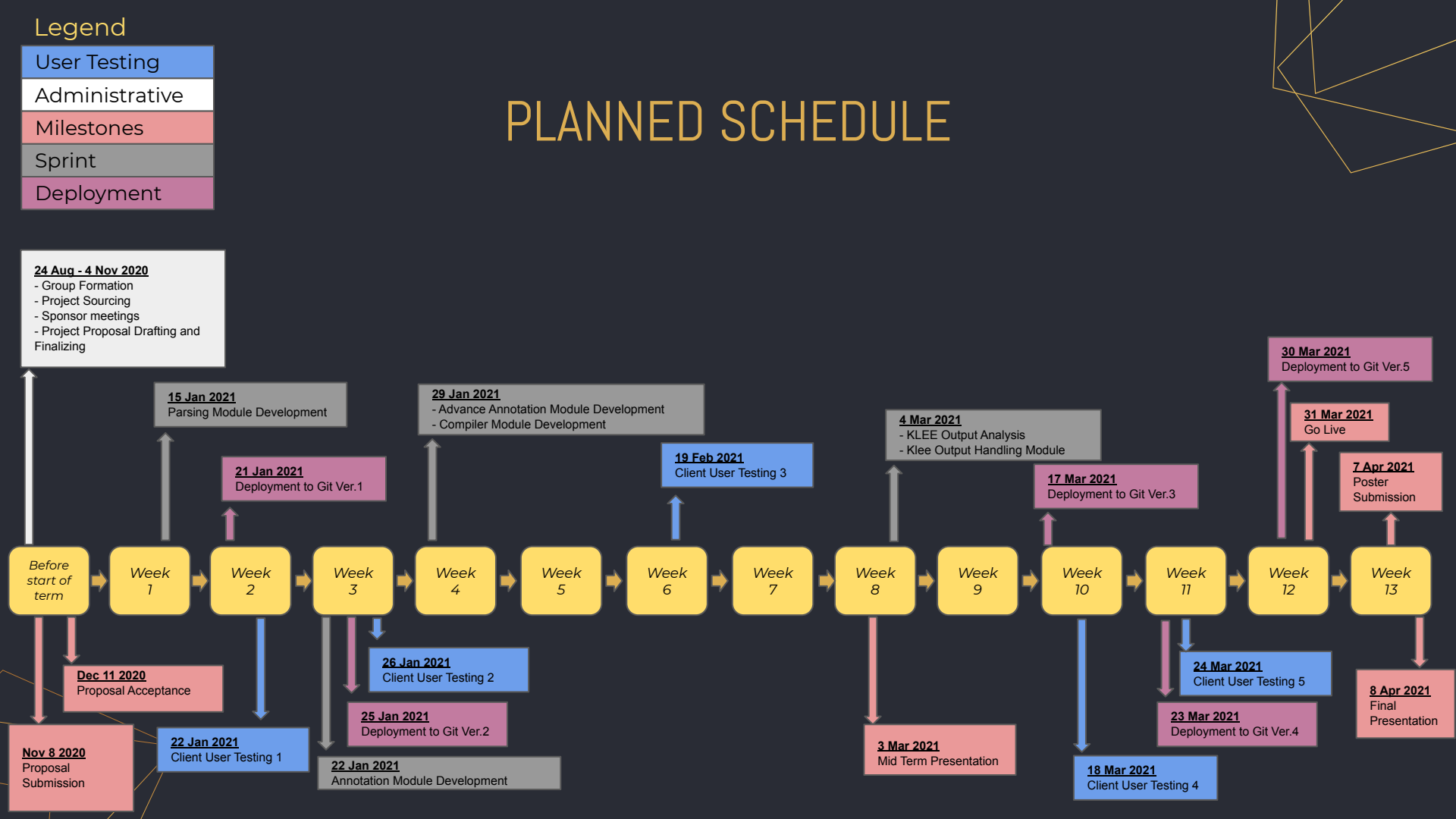
- Regular Regression testing is conducted
- Modular design ease of review for debugging

MEETINGS

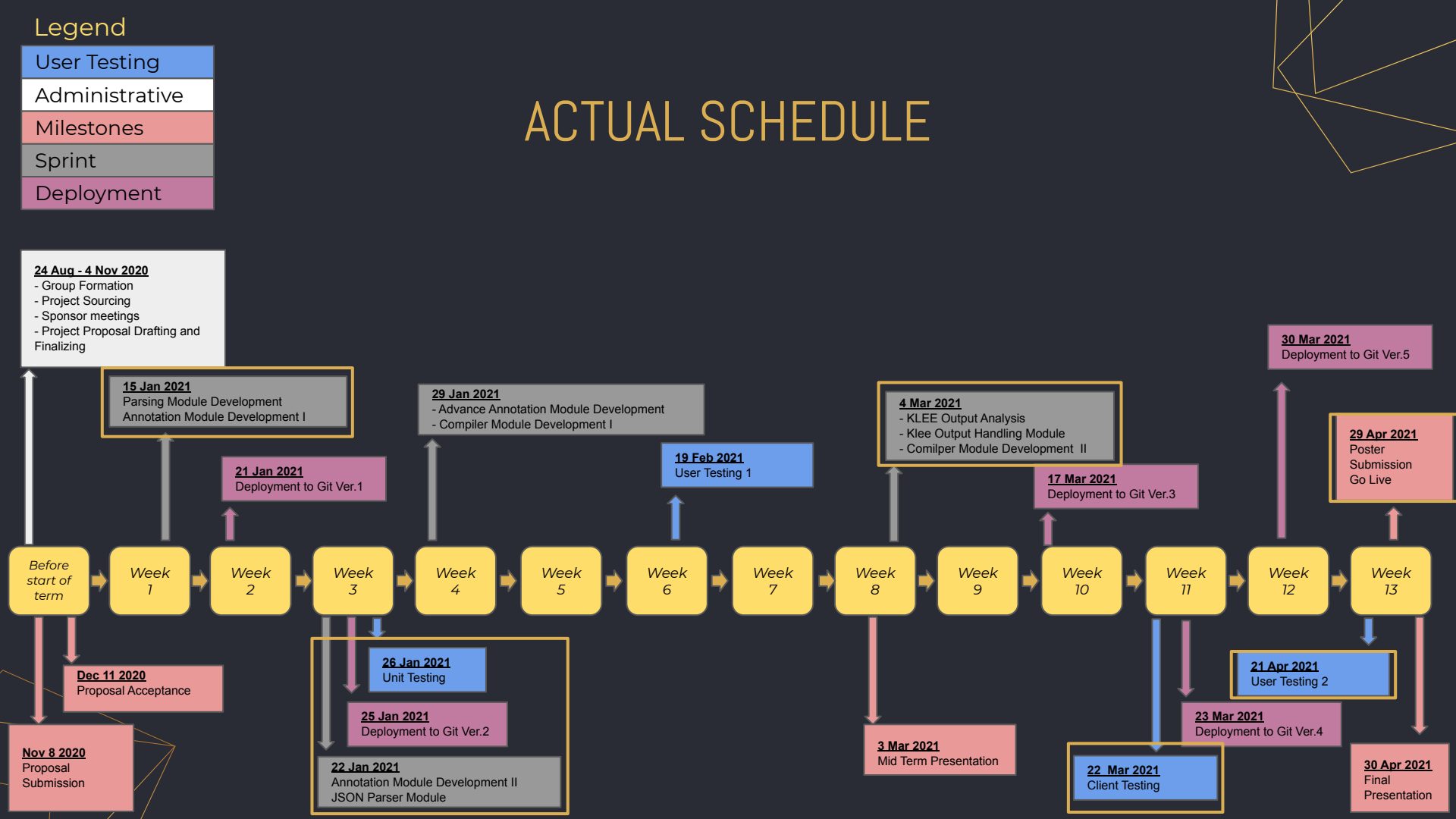
30	9	5
INTERNAL	SPONSOR	SUPERVISOR



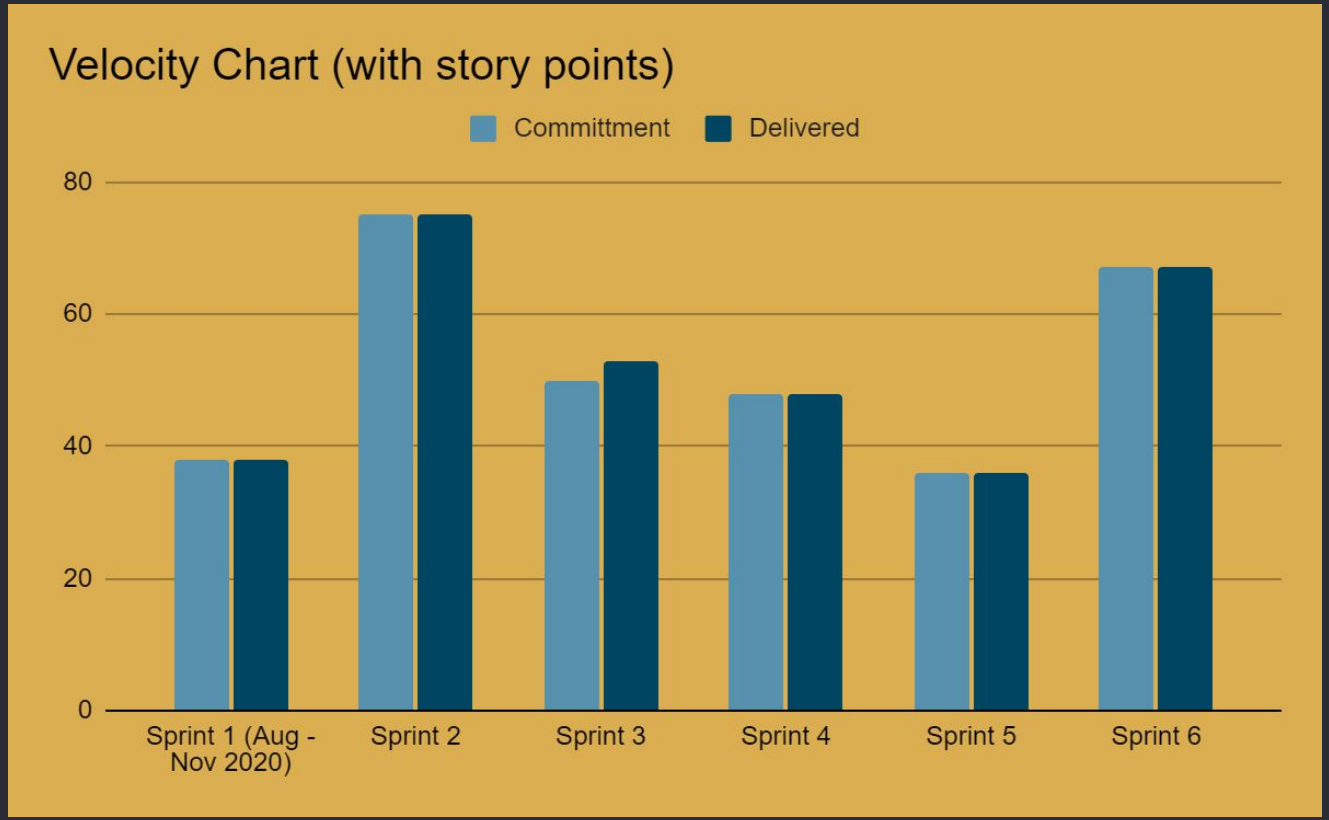
PLANNED SCHEDULE



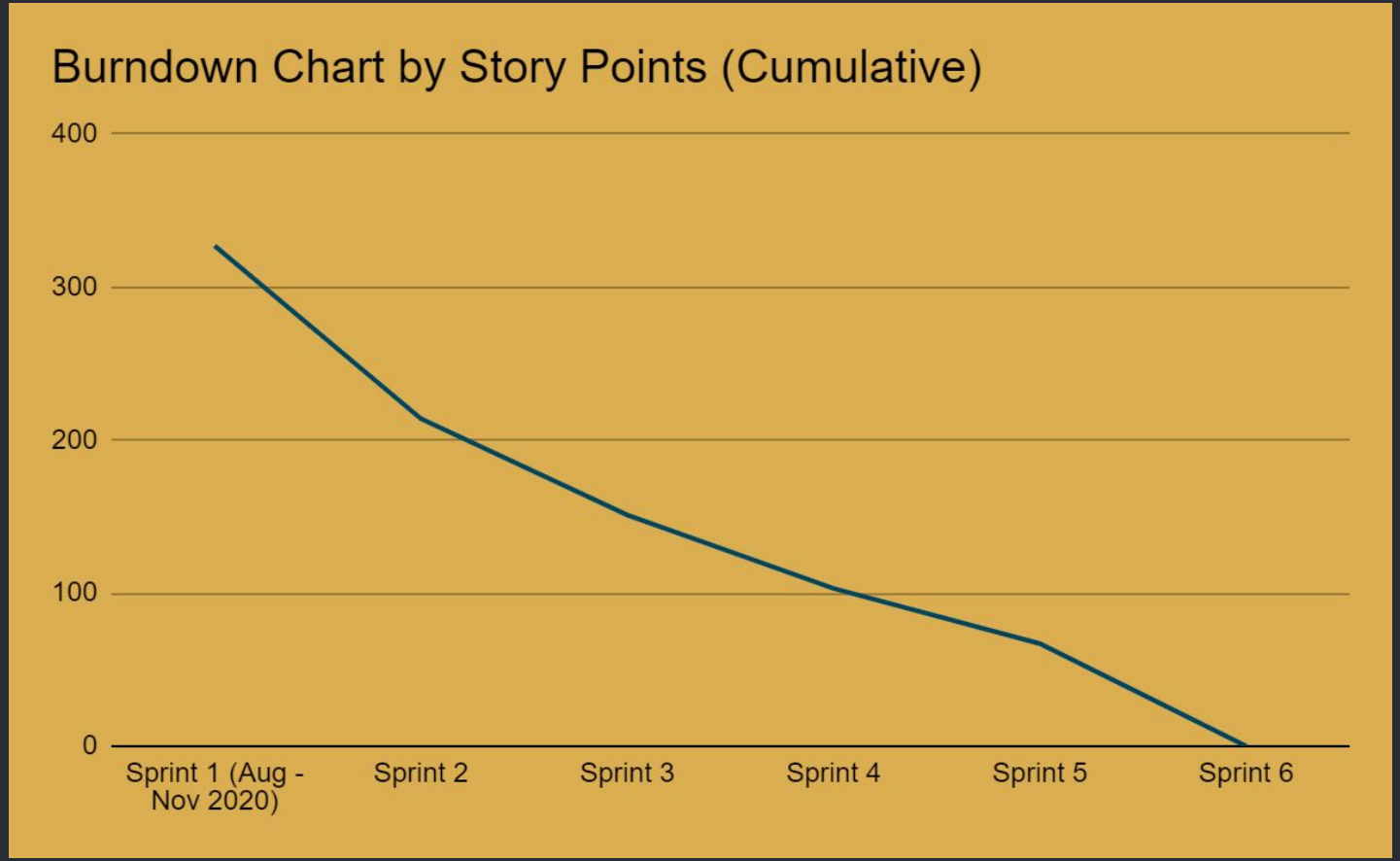
ACTUAL SCHEDULE



METRICS


Velocity Chart (with story points)
Committment
Delivered
80
60
40
20
0
Sprint 1 (Aug - Nov 2020)
Sprint 2
Sprint 3
Sprint 4
Sprint 5
Sprint 6

METRICS


Burndown Chart by Story Points (Cumulative)
400
300
200
100
0
Sprint 1 (Aug - Nov 2020)
Sprint 2
Sprint 3
Sprint 4
Sprint 5
Sprint 6

CHANGE MANAGEMENT

S/N	Sprint No.	Date	Change	Proposed by	Description	Action	Status
8	5	22/3/21	User Testing 4 converted to Client Testing with project sponsor	KLEENOTATION () and Sponsor	Upon discussion with sponsor, and with the constraints of the code, Client Testing was decided upon as the best choice	Client Testing conducted in place of User Testing 4	Closed
9	5	24/3/21	User Testing 4 shifted back	KLEENOTATION ()	Planned functionalities need extra time to be implemented into KLEENOTATION() is not ready on planned date	Client User Testing 4 shifted back	Closed
10	6	1/4/21	Utilisation of cb-multios to compile DARPA codes	KLEENOTATION ()	Compiling methods for POC codes do not work on	Implement cb-multios in KLEENOTATION()	Closed
11	6	1/4/21	Creation of graphical user interface (GUI)	KLEENOTATION () Supervisor	To further speed up the process of writing inputs to the tool, a GUI was created for intuitive display	Creation of GUI	Closed

CHANGE MANAGEMENT

S/N	Sprint No.	Date	Change	Proposed by	Description	Action	Status
12	6	7/4/21	Scope changed from initial proposal	KLEENOTATION () and Sponsor	Upon seeing the results of KLEE, we determined that it was not economical to analyse the accuracy of KLEE's ability to identify bugs from DARPA repository.	Team to focus on building UI and remediation steps instead	Closed
13	6	16/4/21	Separate modules for running DARPA codes	KLEENOTATION ()	Specific requirements are needed to run DARPA codes, therefore some of the modules are modified separately to cater for this	Parser, Annotate, Compiler for DARPA codes created	Closed
14	6	21/4/21	User Testing 2 conducted	KLEENOTATION ()	After the code has been finalised and the modules completed, User Testing 2 was conducted	User Testing 2 conducted	Closed

RISK MANAGEMENT

Risk Type	Risk Event	Likelihood	Impact	Mitigation
Project Management Risk	Time taken to complete a task may exceed the time planned	Low	Medium	• Use proper scheduling to ensure that we have enough buffer time • If not enough time, engage in change management to reschedule
Technical Risk	Little/no knowledge of the inner workings of KLEE, which may lead to more time taken to develop app	Low	Low	• Assign more time to develop modules if necessary • Conduct research on KLEE before embarking on the project • Knowledge sharing within group
Resource Risk	Unexpected case of laptop failure or crash	Low	High	• Code and other documents are backed up in Git and Google Drive, respectively
Client Management	Client may wish to add certain extra functionalities at a later point	Low	Medium	• Discuss with the sponsor to ensure he understands the time taken may increase • Engage in change management if more time is needed to reschedule

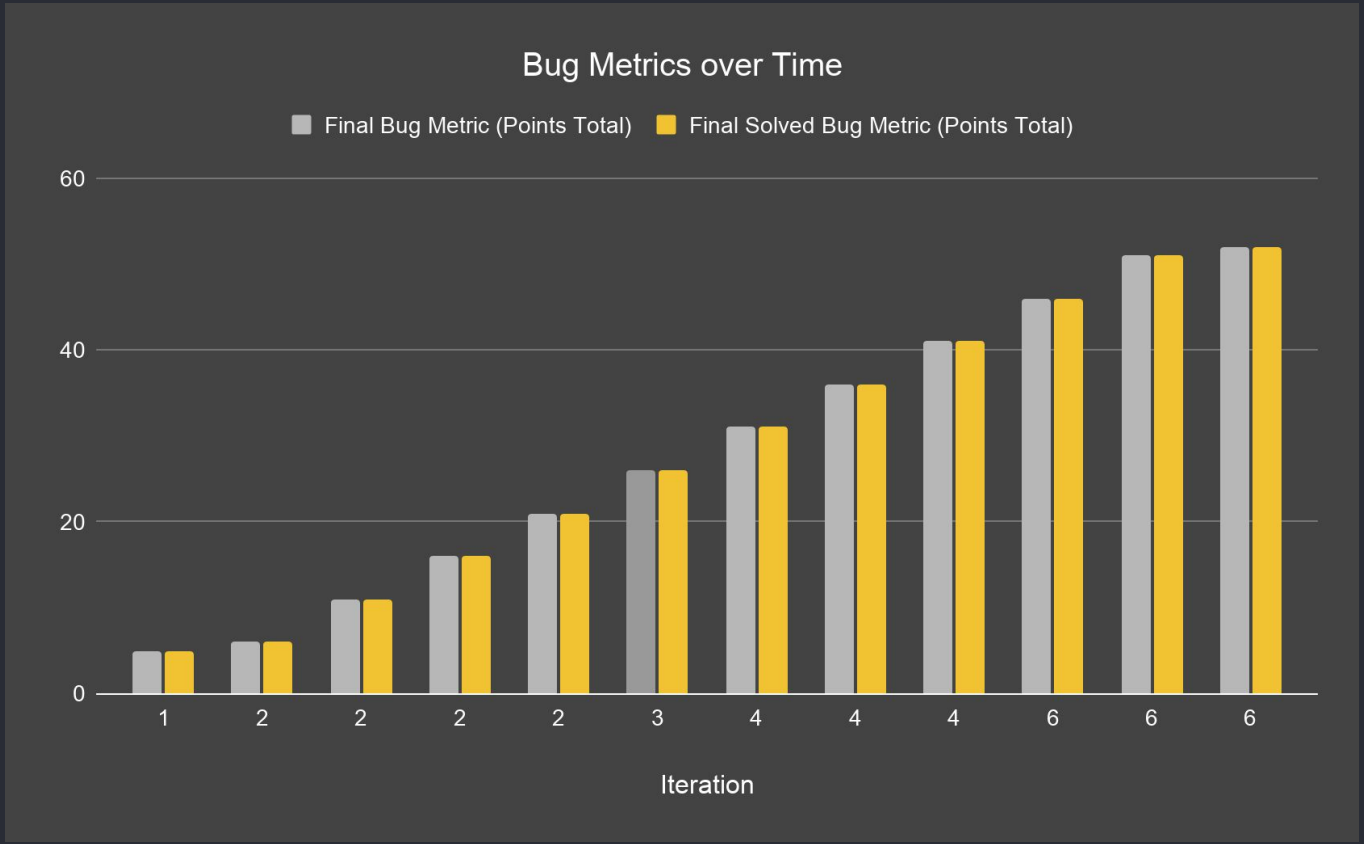
BUG TRACKING

Severity	Description
Low Impact (1 points)	Unimportant. Typo error or small user interface alignment issues.
High Impact (5 points)	The system runs. However, some non-critical functionalities are not working.
Critical Impact(10 points)	The system is down or is un-usable after a short period of time. We have to fix the bugs to continue.
Points in Sprint	**Action**
Points < 10	Use the planned debugging time in the iteration.
Points >= 10	Stop current development and resolve the bug immediately. Project Manager reschedules the project.

BUG TRACKING

Bug Metrics										
S/N	Sprint No.	Function	Description	Severity	Points	Mitgation	Discovered On	Resolved On	Final Bug Metric (Points Total)	Final Solved Bug Metric (Points Total)
1	1	Parser	Parser cannot handle fake headers (required process to generate AST)	High	5	Add functionality to handle fake headers	22-1-2021	27-1-2021	5	5
2	2	Parser	Parser cannot handle JSON formatted inputs	Low	1	Add JSON parsing using python library	27-1-2021	27-1-2021	6	6
3	2	Parser	Array type variables could not be located	High	5	Use search to find array declarations	3-2-2021	3-2-2021	11	11
4	2	Annotate	"&" value did not change when array was being made symbolic	High	5	Use conditional code to assign & only when needed	5-2-2021	5-2-2021	16	16
5	2	Annotate	Error thrown by KLEE when handling scanf functions within C code	High	5	Comment out lines in C program which requires user input	12-2-2021	15-2-2021	21	21
6	3	Klee Utilisation	Issue with os.system() running commands. Does not run command line as per string assigned	High	5	Use subprocess.run()	18-2-2021	18-2-2021	26	26

BUG TRACKING



VALUE TO SPONSOR

- Communication
  - Provided consistent updates to sponsor (meetings & emails)
    - Updated sponsor on progress
    - Gather feedback
    - Make changes (change management) if necessary
- Time-saving
  - Saved sponsor time by automating his processes
- Accuracy
  - Code meets the functional requirements of sponsor

VALUE TO SPONSOR

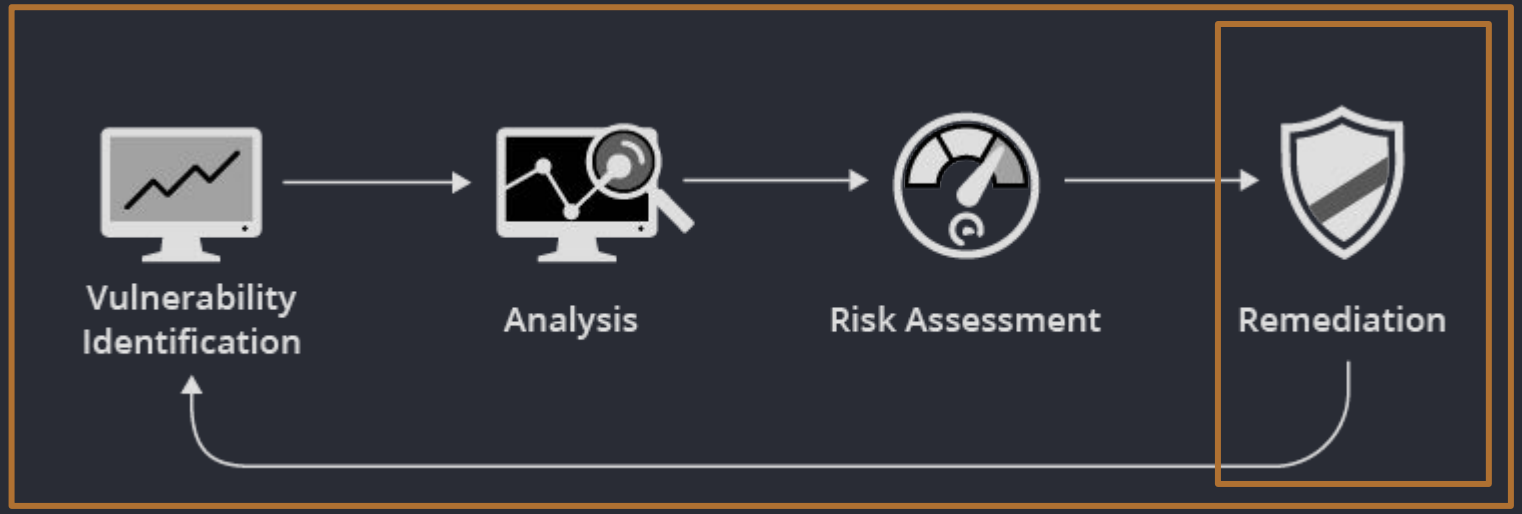
- Extending current functionalities of KLEE
  - Alert user to possible vulnerabilities based on errors discovered
    - Assess possible risks based on predefined conditions
  - Decrease scope of investigation required for each C program
    - User only needs to investigate high-risk lines of code

6

CONCLUSION

Summary of Project
Future work: Growth & Continuity

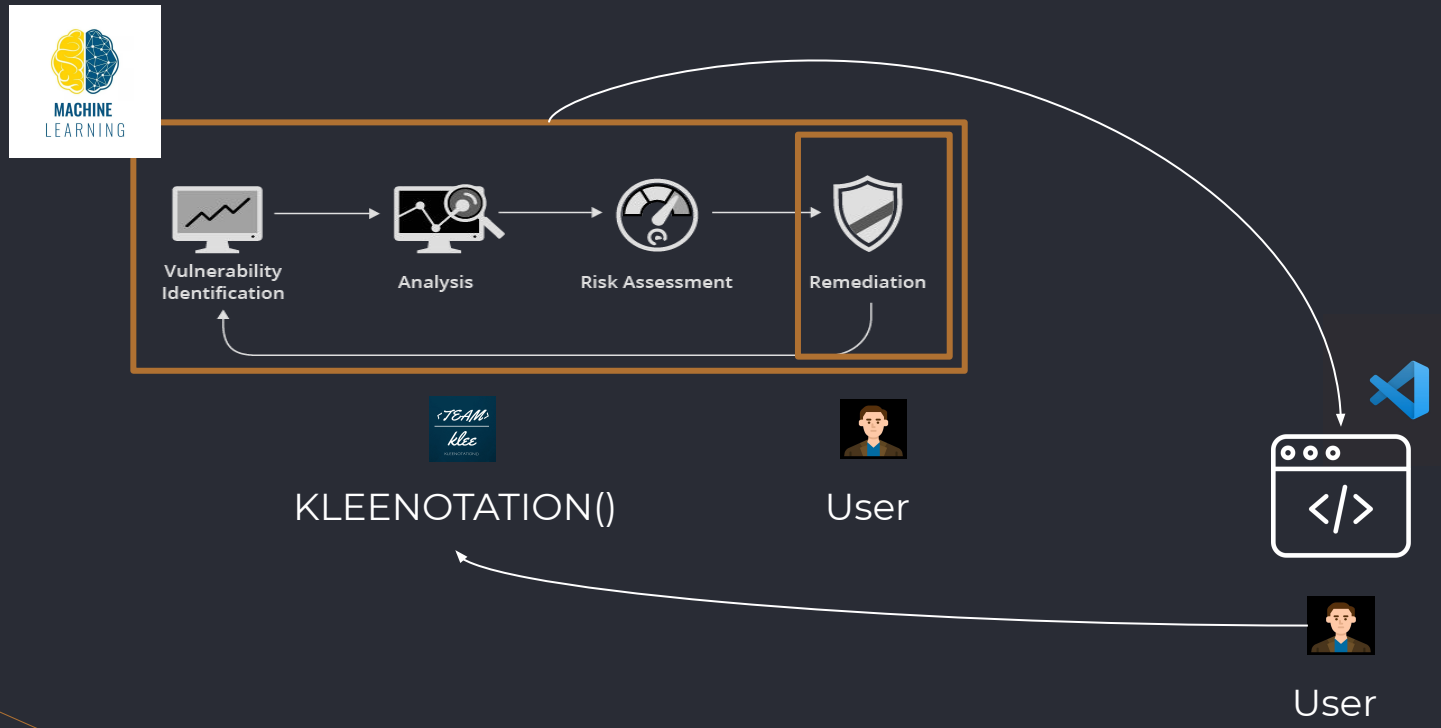
SOLUTION OVERVIEW



KLEENOTATION() automation of transformation

User

FUTURE WORK: SECURE DEVELOPMENT



DEMO

Special Thanks

Our Sponsor
Prof. Jiang Lingxiao
(lxjiang@smu.edu.sg)

SMU PhD Student
Tu Haoxin
(haoxintu.2020@phdcs.smu.edu.sg)

Q & A