Semi-Automated Detection of Sanitization, Authentication and Declassification Errors in UML State Charts

Md Adnan Rabbi
Department of Computer Science
Technical University of Munich, Germany

Agenda

- Introduction
- Background
- Challenges and Annotation Language Extension
- Implementation
- Experiments
- Conclusion and Future Work

Introduction

- **What do we want to achieve in this work?**
  - Develop a tool which can help to detect (sanitization, declassification and authentication) (*) errors during software design phase.
- **What is the problem with detection of (*) errors during SW Dev.?**
  - Usually (*) are addressed during coding phase.
- **What other solution exist?**
  - To the best of our knowledge there are no other tools which can check (*) during design.
- **Where do these tools lack?**
  - They don’t have support for checking (*) in models.
- **What is our insight?**
  - (*) can be addressed using UML state charts, annotation language and other tools.
- **What are our contributions?**
  - Ann. Lang. extensions, C code gen., 3 info. flow checkers, UML seq. diagram gen., IEEE QRS-C’15 Publication

Agenda

- Introduction
- **Background**
- Challenges and Annotation Language Extension
- Implementation
- Experiments
- Conclusion and Future Work

Background I

Sanitize User Input:

- Process of removing information (forbidden characters, sensitive, confidential etc.) from user input.
- In order to protect SQL injection, cross-site scripting (XSS) attacks sanitization can be used.

Background II

Authenticate User Access:

- Authentication is the mechanism which confirms the identity of users trying to access a system.
- Generally, this is handled by passing a key with each request.
- Often called an access token, user verification using user id and password.

Background III

Declassify Confidential Information:

- Enabling information flow controls with expressive information release or declassification policies.
- Lowering the security classification of selected information.
- Sometimes encrypting/decrypting policies.

Background IV

- Information flow propagation theory can be used to detect these types of bugs.
- The UML model is annotated with information flow annotations.
- The UML model is converted to code containing annotations.
- A control flow graph is generated from the generated source code.
- Paths in the control flow graph are checked in order to detect information flow error based on the previous added annotations.

Agenda

- Introduction
- Background Information
- **Challenges and Annotation Language Extension**
- Implementation
- Experiments
- Conclusion and Future Work

Challenges and Annotation Language Extension I

- Detect information flow bugs in UML state charts and C code.
- An annotation language which can be used to annotate UML state charts and code.
- Information flow restrictions which can be added during two software development phases (design and coding).

Challenges and Annotation Language Extension II

Annotation Type	Annotation Tag	Description
@function	**Authentication,declassification, sanitization**	authenticate, declassifies and sanitizes information
@parameter	**authenticated H/L declassified H/L sanitized H/L**	authenticated , declassified and sanitized with High/Low tags
@variable	**confidential H/L, source H/L**	confidential and source with High/Low tags

Challenges and Annotation Language Extension III

Annotation Language Design Process:

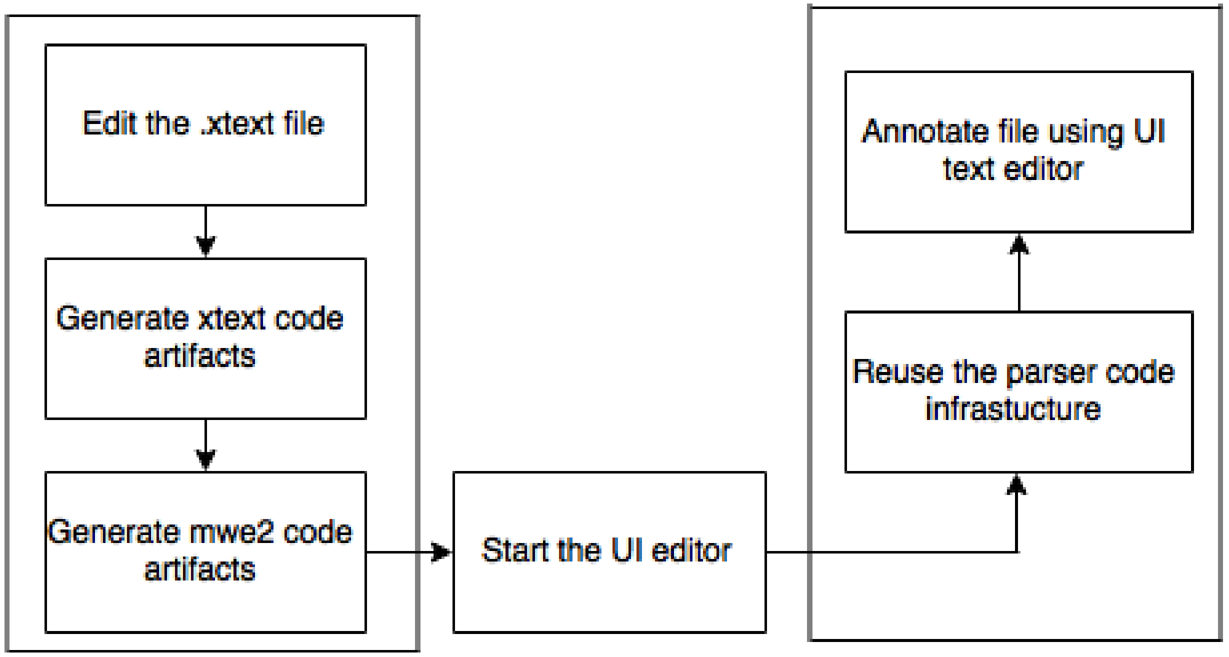

Edit the .xtext file
Generate xtext code artifacts
Generate mwe2 code artifacts
Start the UI editor
Reuse the parser code infrastucture
Annotate file using UI text editor

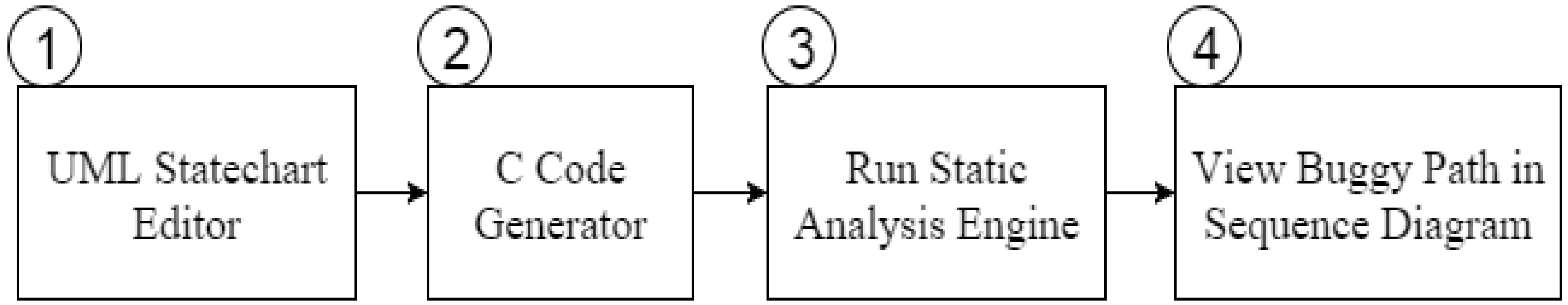
Figure 1: Annotation language design process

Agenda

- Introduction
- Background Information
- Challenges and Annotation Language Extension
- **Implementation**
- Experiments
- Conclusion and Future Work

Implementation I

Overview of System Architecture


1
UML Statechart Editor
2
C Code Generator
3
Run Static Analysis Engine
4
View Buggy Path in Sequence Diagram

- UML Statechart Editor
- C Code Generator
- Static Analysis Engine (three new checkers added)
- Sequence Diagram Generator

Implementation II

Tools and Technologies

- Eclipse Xtext
- Eclipse Xtend
- YAKINDU SCT Editor
- EMF (Eclipse Modeling Framework)

Agenda

- Introduction
- Background Information
- Challenges and Annotation Language Extension
- Implementation
- **Experiments**
- Conclusion and Future Work

Experiments I

UML StateChart Model:

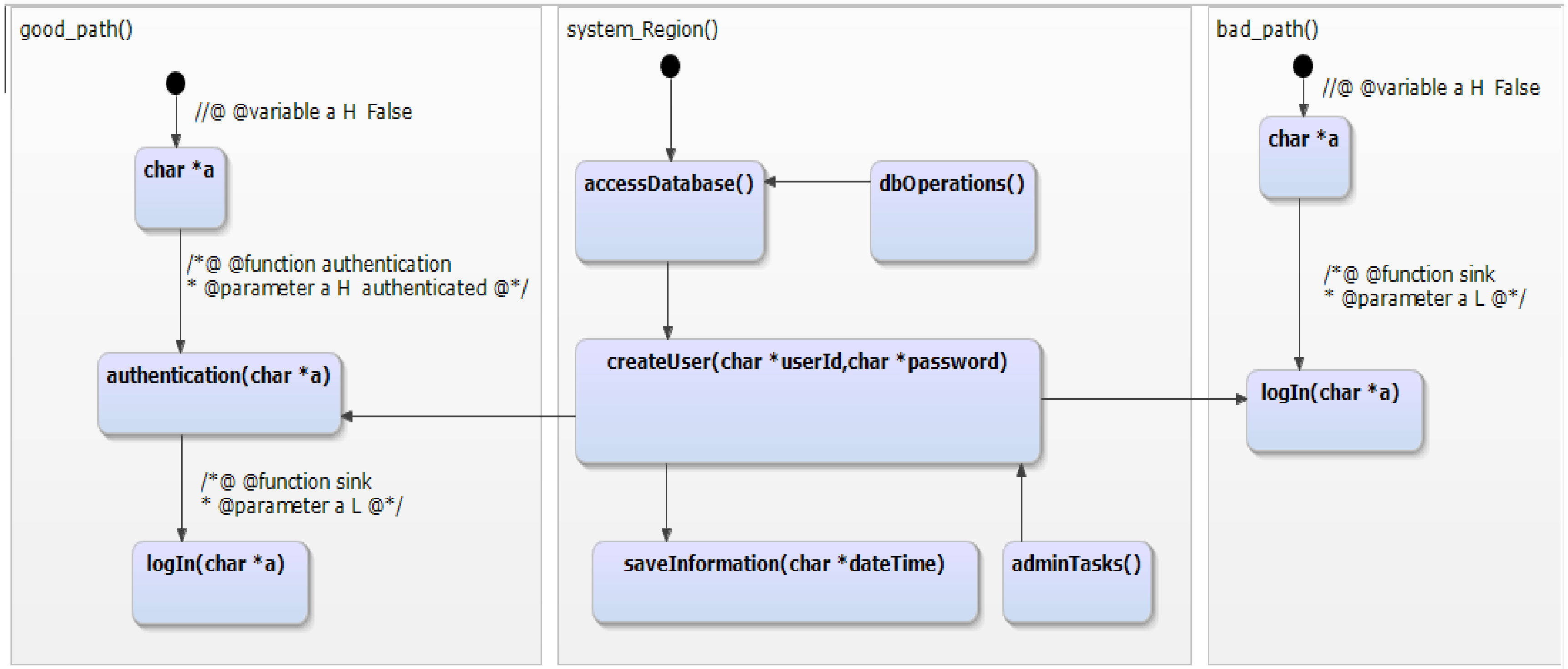


Figure 2: Authentication scenario (CWE-306) statechart model

Experiments II

UML StateChart Model:

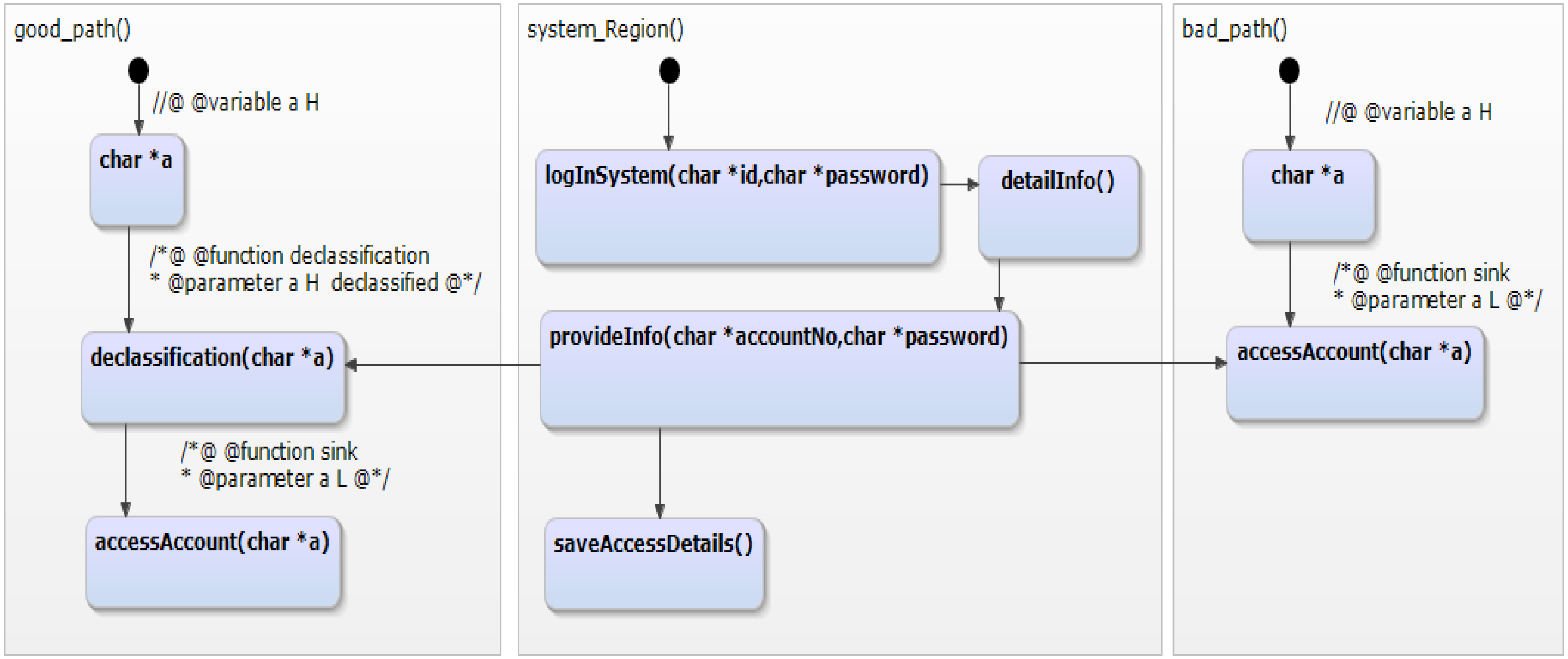


Figure 3: Declassification scenario statechart model

Experiments III

UML StateChart Model:

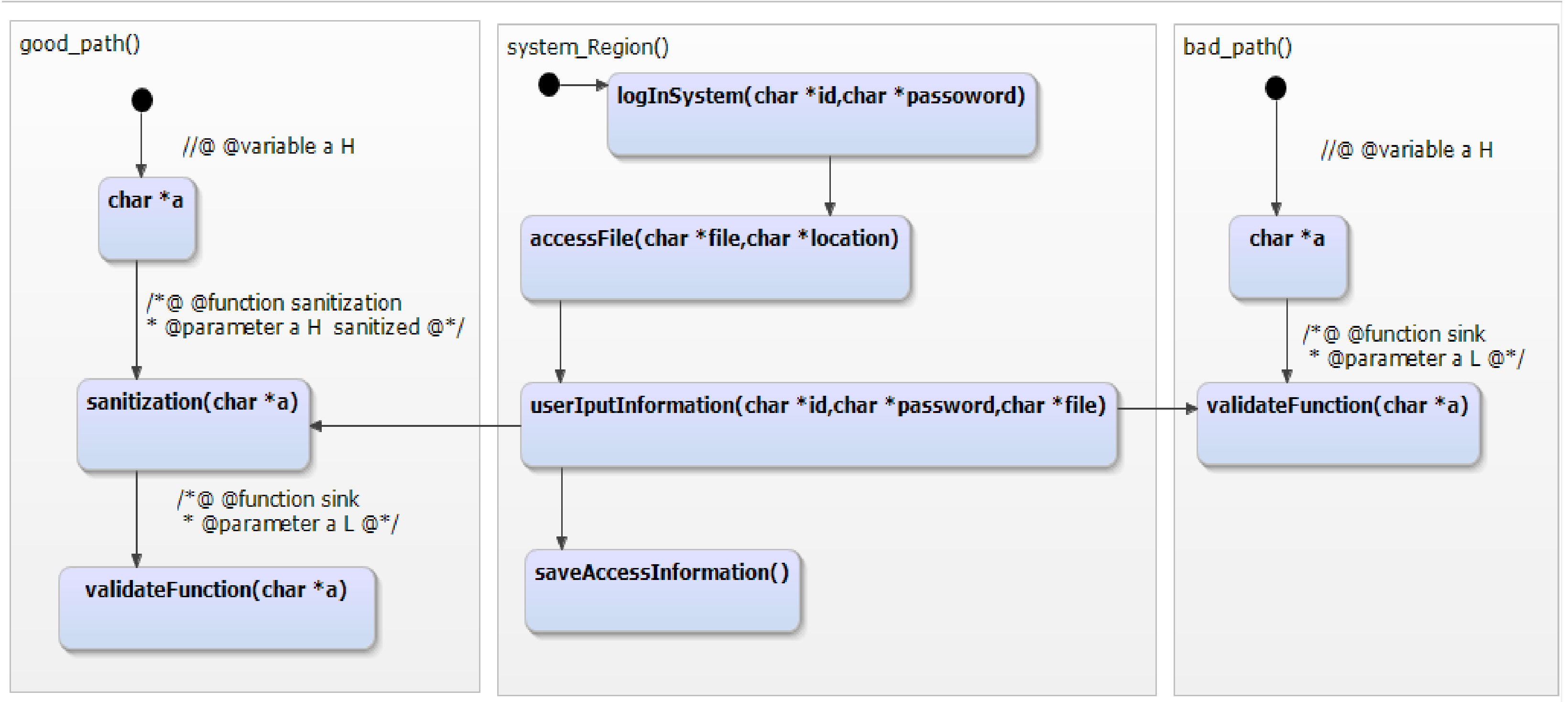

good_path()
//@ @variable a H
char *a
/*@ @function sanitization
* @parameter a H sanitized @*/
sanitization(char *a)
/*@ @function sink
* @parameter a L @*/
validateFunction(char *a)
system_Region()
logInSystem(char *id,char *passoword)
accessFile(char *file,char *location)
userIputInformation(char *id,char *password,char *file)
saveAccessInformation()
bad_path()
//@ @variable a H
char *a
/*@ @function sink
* @parameter a L @*/
validateFunction(char *a)

Figure 4: Sanitization scenario (CWE-78) statechart model

Experiments IV

C Code Generator:

- Using Eclipse Xtend.
- Extended YAKINDU SCT Editor.

Generated C code files sample:

C header file (.h file) contents:

```
/*@ @function sink
* @parameter  a L @*/;
void logIn(char *a)
… … … … … … … …
```

Experiments V

C source file (.c file) contents:

void good_path(){

 logIn(a) ;

//@ **@variable** a **H** False ;

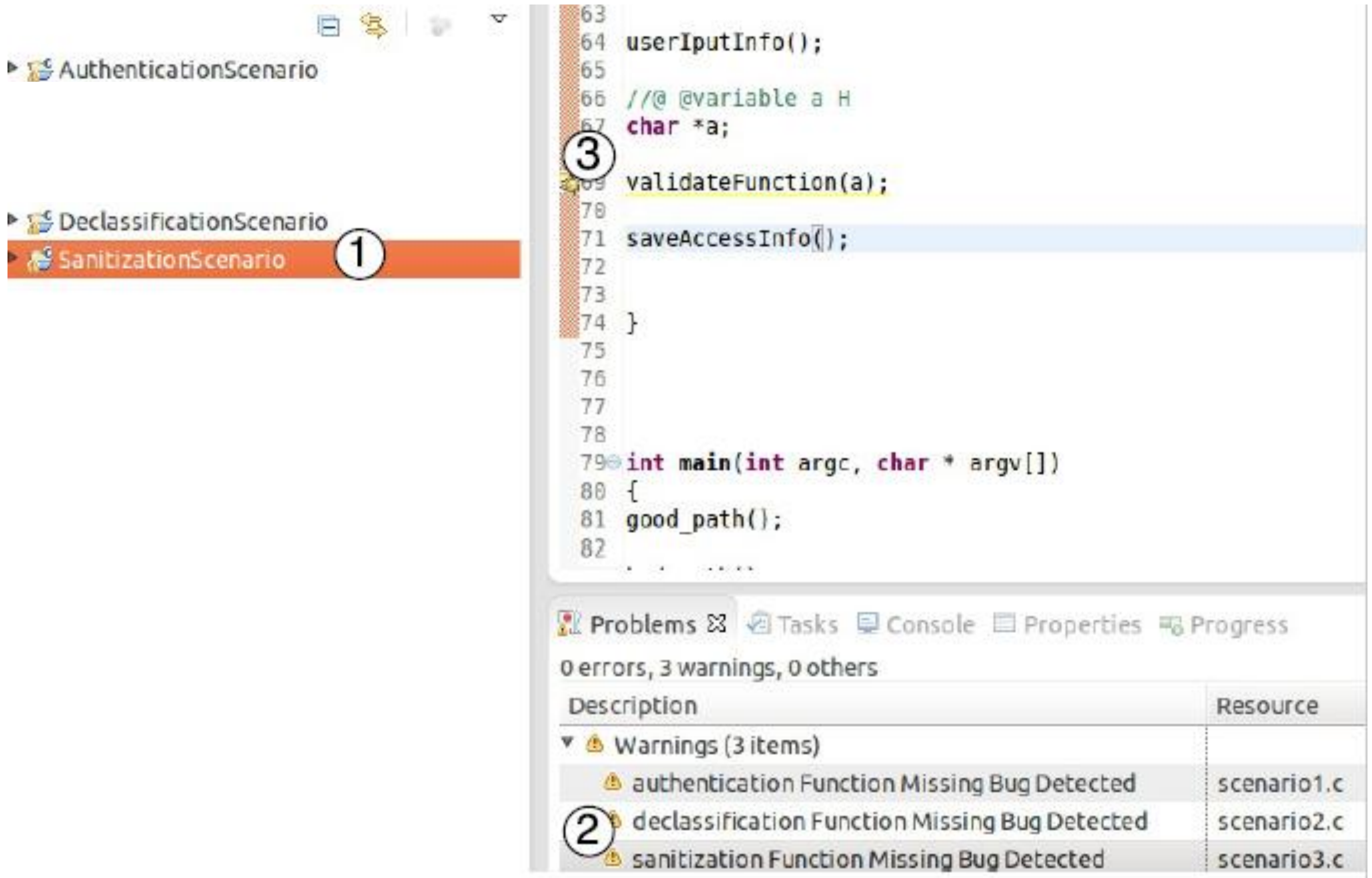
char *a;

authentication(a);

saveInformation(dateTime);

… … … … … … … }

Experiments VI



```
userIputInfo();

//@ @variable a H
char *a;

validateFunction(a);

saveAccessInfo();


}




int main(int argc, char * argv[])
{
good_path();
```

0 errors, 3 warnings, 0 others

Description	Resource
Warnings (3 items)	
authentication Function Missing Bug Detected	scenario1.c
declassification Function Missing Bug Detected	scenario2.c
sanitization Function Missing Bug Detected	scenario3.c

Figure 5: Bug Detection with Static Analysis Engine

Experiments VII

Sequence Diagram Generator:

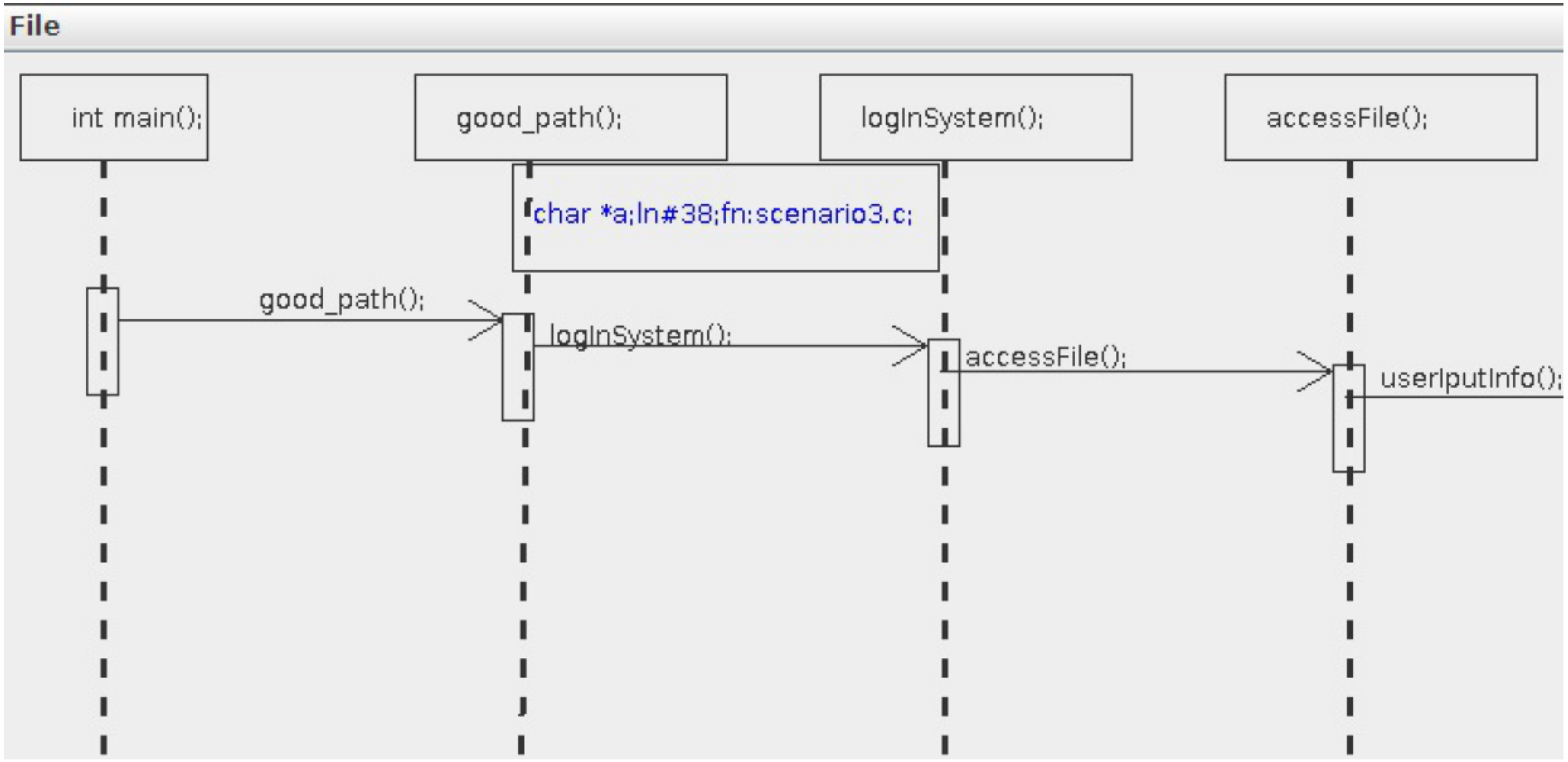


Figure 6: View Buggy Path in UML Sequence Diagram

Agenda

Conclusion and Future Work

Conclusion:

- A keyword-based annotation language was developed that can be used for annotating UML state charts and C code.
- Detect information flow bugs automatically and it is applicable to real life scenarios.
- In order to detect authentication, declassification and sanitization errors, our tool can be used in the design and coding phase of software development.

Conclusion and Future Work

Future Work:

- Our tool can be extended for source code editor as a pop-up window based proposal editor.
- The editor can be extended for other types of diagrams.
- The annotation lang. can be extended in order to deal with other scenarios.

Q/A?

Thank You ALL !!!