FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis

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

Md Adnan Rabbi

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis

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

Halbautomatische Erkennung von Sanitisierungs,Authenifizierungs und Deklassifierungsfehlern in UML-Zusantsdiagrammen.

Author: Md Adnan Rabbi

Supervisor: Prof. Dr. Claudia Eckert

Advisor: MSc. Paul-loan Muntean

Date: November 15, 2015

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 16. August 2015 Md Adnan Rabbi

Acknowledgments

If someone contributed to the thesis... might be good to thank them here.

Abstract

An abstracts abstracts the thesis!

Contents

Outline of the Thesis

Part I: Introduction and Theory

CHAPTER 1: INTRODUCTION

This chapter presents an overview of the thesis and it purpose. Furthermore, it will discuss the sense of life in a very general approach.

CHAPTER 2: BACKGROUND INFORMATION

This chapter describes the background and the essential theory to establish the research.

Part II: Implementation and Analysis

CHAPTER 3: CHALLENGES AND ANNOTATION LANGUAGE EXTENSION

This chapter presents the challenges and annotation language extension for the system.

CHAPTER 4: IMPLEMENTATION

This chapter presents the implementation of the system.

CHAPTER 5: EXPERIMENTS

This chapter presents the different application area of the system.

Part III: Conclusion and Future Work

CHAPTER 6: CONCLUSION AND FUTURE WORK

This chapter presents the conclusion of the whole research along with future work intentions.

Part I.

Introduction and Theory

1. Introduction

The detection of information flow vulnerabilities uses dynamic analysis techniques , static analysis techniques and hybrid techniques which combine static and dynamic approaches. The static techniques need to know when to use sanitization , declassification and authentication functions. A solution for tagging sanitization, declassification and authentication in source code is based on libraries which contain all needed annotations attached to function declarations. This approach plays an important role mainly for static analysis bug detection techniques where the information available during program run-time is not available nor the interaction with the environment can be fully simulated. Extended Static Checking (ESC) is a promising research area which tries to cope with the shortage of not having the program run-time information. During extended static analysis additional information is provided to the static analysis process. This information can be used to define trust boundaries and tag variables. Textual annotations are usually manually added by the user in source code. At the same time annotations can be automatically generated and inserted into source code . ESC can be used to eliminate bugs in a late stage of the software project when code development is finished. Tagging and checking for information exposure bugs during the design phase would eliminate the potential of implementing software bugs which can only be removed very costly after wards. Thus security concerns should be enforced into source code right after the conceptual phase of the project. The paper presents five challenges concerning ESC. The last challenge reports of the annotation as being a very time consuming burden and is therefore disliked by some programming teams. The authors argue about the fact that annotations can cover design decisions and enhance the quality of source code. We argue that annotations are necessary in order to do ESC and the user needs a kind of assistance tool that helps selecting the suited annotation based on the current context. Thus the annotation burden needed for learning and applying the language should be reduced. At the same time adding annotations to reusable code libraries reduces even more the annotation burden since libraries can be reused, shared and changed by software development teams.

1.1. Latex Introduction

There is no need for a latex introduction since there is plenty of literature out there.

2. Background Information

2.1. Sanitization

Sanitization is the process of removing sensitive information from a document or other message or sometimes encrypting messages, so that the document may be distributed to a broader audience.Sometimes sanitization can be called as an operation that ensures that user input can be safely used in an SQL query. Some basic purpose of sanitization are given below:

- To identify the set of parameters and global variables which must be sanitized before calling functions.
- It is acceptable to first pass the untrusted user input through a trusted sanitization function.
- Any user input data must flow through a sanitization function before it flows into a SQL query.
- Confidential data needs to be cleansed to avoid information leaks.
- Most paths that go from a source to a sink pass through a sanitizer.
- Developers typically define a small number of sanitization functions or use ones supplied in libraries.

2.2. Declassification

Information security has a challenge to address: enabling information flow controls with expressive information release (or declassification) policies. In a scenario of systems that operate on data with different sensitivity levels, the goal is to provide security assurance via restricting the information flow within the system.
To declassify information means lowering the security classification of selected information. Sabelfeld and Sands [1] identify four different dimensions of declassification, what is declassified, who is able to declassify, where the declassification occurs and when the declassification takes place.

2.3. Authentication

Authentication is the mechanism which confirms the identity of users trying to access a system. For a user to be granted access to a resource, they must first prove that they are who they claim to be. Generally this is handled by passing a key with each request (often called an access token, User verification using user id and password). The system or server verifies that the access token or user id and password is genuine, that the user does indeed have the required privileges to access the requested resource and only then is the requset granted. Also authentication can be defined as it is the process by which the system validates a user's logon information. A user's name and password are compared to an authorized list and if the system detects a match then access is granted to the extent specified in the permission list for that user.

Part II.

The Second Part

3. Challenges and Annotation Language Extension

3.1. Challenges and Idea

Previous annotation language grammer has been extended more to detect implicit and explicit information flow bugs in UML state charts and C code. The purpose of the same annotation language can be used to add information flow constraints to UML state charts and code in order to detect information flow errors.

The challenge was addressed by extending the annotation language containing textual annotations which can be used to annotate source code and UML state charts which are backward compatible.The single-line annotations have the same as previous consisting start tag "//@" and the multi-line annotations have the start tag "/*@" and the end tag "@*/" .

Some challenges throughout the approach are- converting textual comments into annotations objects, introducing syntactically correct annotations into files, how to use the same annotation language in order to annotate UML state charts and source code, dealing with scattered annotations and attaching annotations to the right function declaration or variable.

3.2. Annotation Language Tags

4. Implementation

5. Experiments

6. Related Work

The detection of information flow errors [31] can be addressed with dynamic analysis techniques [2], [16], [48], static analysis techniques [17], [41], [51], [58], [60] (similar to our approach with respect to static analysis of code and tracking of data information flow) and hybrid techniques which combine static and dynamic approaches [38]. Also, extended static checking [10] (ESC) is a promising research area which tries to cope with the shortage of not having certain program run-time information. The static code analysis techniques need to know which parts of the code are: sinks, sources and which variables should be tagged. A solution for tagging these elements in source code is based on a pre-annotated library which contains all the needed annotations attached to function declarations. Leino [27] reports about the annotation burden as being very time consuming and disliked by some programming teams. There are many annotation languages proposed until now for extending the C type system [9], [13], [29], [30], [57] to be used during run-time as a new language run-time for PHP and Python [61] to annotate function interfaces [13], [29], [57], to annotate models in order to detect information flow bugs [24] to annotate source code files [46], [47], [56] or to annotate control flows [13], [15], [29]. The studies rely on manually written annotations while our annotation language is integrated into two editors which are be used to annotate UML state charts and C code by selecting annotations from a list and without the need to memorize a new annotation language. The following annotation languages have made significant impact: Microsoftś SAL annotations [29] helped to detect more than 1000 potential security vulnerabilities in Windows code [3]. In addition, several other annotation languages including FlowCaml [50], Jif [7], Fable [55], AURA [22] and FINE [54] express information flow related concerns. Recently taint modes integrated in programming languages as Camlbased FlowCaml [52], Ada-based SPARK Examiner [5] and the scripting. However, none of these annotation and programming languages have support for introducing information flow restrictions in both models and the source code. Splint [14], Flawfinder [59] and Cqual [49] are used to detect information flow bugs in source code and come with comprehensive user manuals describing how the annotation language can be used in order to annotate source code. iFlow [24] is used for detecting information flow bugs in models and is based on modeling dynamic behavior of the application using UML sequence diagrams and translating them into code by analyzing it with JOANA [25]. In comparison with our approach these tools do not use the same annotation language for annotating UML models and code. Thus, a user has to learn to use two annotation languages which can be perceived to be a high burden in some scenarios. UMLSec [23] is a model-driven approach that allows the development of secure applications with UML. Compared with our approach, UMLSec does neither include automatic code generation nor the annotations can be used for automated constraints checking. Heldal et al. [18], [19] introduced an UML profile that incorporates a decentralized label model [40] into the UML. It allows the annotation of UML artifacts with Jif [42] labels in order to generate Jif code from the UML model automatically. However, the Jif-style annotation already proved to be non-trivial on the code level [45], while [19] notes that the actual automatic Jif code generation is still future work. These approaches can not be used to annotate both UML models and code. Moreover, these approaches lack of tools for automated checking of previously imposed constraints.

7. Conclusion and Future Work

A keyword-based annotation language that can be used out of the box for annotating UML state charts and C code in two software development phases by providing two editors for inserting security annotations in order to detect information flow bugs automatically. It's evaluated on some sample programs and showed that this approach is applicable to real life scenarios.

It's a light-weight annotation language usable for specifying information flow security constraints which can be used in the design and coding phase in order to detect information flow bugs.

In future it can be extended for source code editor as a pop-up window based proposal editor used to add/retrieve annotation to/from a library. The definition of new language annotation tags should be possible from the same window by providing two running modes (language extension mode and annotation mode). The envisaged result is to reduce the gap between annotations insertion/retrieval and the definition of new language tags. This would help to create personalized annotated libraries which can be collaboratively annotated if needed.

Appendix

A. Detailed Descriptions

Here come the details that are not supposed to be in the regular text.

Bibliography

[1] Andrei Sabelfeld and David Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.