DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Impact factors for severity assessment of bugs discovered via compositional symbolic execution

Ricardo Nales

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Impact factors for severity assessment of bugs discovered via compositional symbolic execution

Einflussfaktoren zur Bewertung der Severity von Programmfehlern, welche mit Compositional Symbolic Execution entdeckt wurden

Author:	Ricardo Nales
Supervisor:	Saahil Ognawala
Advisor:	Profr. Dr. Alexander Pretschner
Submission Date:	Submission date

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Submission date Ricardo Nales

Acknowledgments

To my supervisor, for always guiding me, and pointing me in the right direction. To my friends, for always helping, believing and encouraging me. And to my family, without them none of this would have been possible. My most heartfelt thanks to all of you.

Abstract

Hello

Contents

1 Motivation

1.1 The severity of bugs

Since the inception of informatics, errors in code, be it semmantic or synactic, have existed. These errors have been attributed the name of software bugs.

Most errors arise from simple programming mistakes, and vary from one programing language to another, as well as the field where these programms are being put to use. Reagrdless of how a bug comes to be, a solution needs to be found for it, as well as a test, and a deployment of for this fix.

1.1.1 How are bugs found

A bug can be found in a pleathora of ways: it can be a simple semicolon missing from the code, it could be a typo in one variable, these are now-a-days easy to find since there are IDEs (Integrated Development Environment); or it could be that someone overlooked sanitizing an input, that could lead to buffer-overflow errors during runtime. Whichever the source of a given bug, the approach to find a it is always the same: reproduce this error by supplying the inputs that got the software to an unstable state, or in most cases, crashed it.

1.1.2 Symbolic execution

Symbolic execution is a way of testing a program to determine what inputs allow the software to execute. Instead of using normal variables for testing, an interpreter follows the execution assuming symbolic values as the inputs instead of using concrete values – as per in normal execution – which can take any value for the given type of the input, thus covering the complete range of possible branches the program could take, therefore discovering bugs along its way.

Throughout this thesis MACKE (Modular and Compositional Analysis with KLEE Engine)[Ogn16] has been used to find bugs, and callgraphs of programs to fully analize the source of the bug, and the possible severity of it.

1.1.3 The severity of a bug

A bug has the power to bring a whole system down. They can be a force to be reckoned with, when it has spawned inside a codebase that has built up to millions of lines of code. A fix for it could be trivial, if a developer understands the nature of the bug, and/or has experience with the system, and/or he introduced the bug himself.

But what happens in the normal occation when a bug is found and there is no easy solution at hand? When resources are scarse and we need to prioritize which bug to focus on next so that our system may continue working as inteded? When there are extremely huge time constraints and we can only budget to fix one at a time, because our workforce is limited?

An institution called FIRST.Org, Inc. (FIRST), a US-based non-profit organization, whose mission is to help computer security incident response teams across the world[FIR] asked themselves these same questions back in 2005, when they decided to introduce CVSS, or Common Vulnerability Scoring System[FIR].

Throughout this thesis we introduce ways to find the severity of a specific bug. Depending on several values, and using the aforementioned metric CVSS3 as our basis to rate them, we will give a bug a numeric score, which could allow us to know what bug should be given priority, and which one might be harmless.

List of Figures

Bibliography

[FIR] I. (FIRST.Org. *Common Vulnerability Scoring System v3.0: Specification Document*.

[Ogn16] e. a. Ognawala S. *MACKE: Compositional analysis of low-level vulnerabilities with symbolic execution*. 2016.