TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript
-
Seminar Report

Nico Fechtner

Technical University of Munich
Department of Informatics
Chair for IT Security
Boltzmannstrae 3, 85748 Garching, Germany
nico.fechtner@tum.de

Table of Contents

Abstract. JavaScript is dynamically and weakly typed which makes it possible to write type inconsistent code. This can often lead to bugs. TypeDevil addresses this issue with a mostly dynamic type inconsistency analysis which is able to warn developers about critical type related bugs. An alternative approach would be to use an additional static type system like e.g. TypeScript.

Keywords: Dynamic Type Inconsistency Analysis for JavaScript, TypeDevil, Static Type Systems for JavaScript, TypeScript

1 Introduction

This report is part of the seminar "Common Security Flaws in JavaScript based Applications", which was organized by Paul Muntean from the Chair of IT Security at the Faculty of Informatics of the Technical University of Munich. The seminar took place in the summer semester 2018 and dealt with multiple scientific papers related to JavaScript Security.
I personally took a deeper look at the paper "TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript", published in 2014 by Michael Pradel, Parker Schuh and Koushik Sen. In this report I would like to explain the general problem the paper tries to address, introduce TypeDevil as a possible solution and compare TypeDevil to an alternative approach, namely additional static type systems.

2 Inconsistent Types as the Root Cause of Many Bugs

First of all, I would like to introduce the general problem, TypeDevil wants to solve.
JavaScript has two interesting characteristics we will focus on. On the one hand, JavaScript is dynamically typed. This basically means, that we do not provide any static type annotations to our source code, like you would in statically typed languages, such as e.g. Java, and that types can change during runtime. On the other hand, JavaScript is also weakly typed and thereby very permissive. In order to prevent runtime exceptions, JavaScript performs a lot of automatic type conversions, also known as implicit coercions. Consider the code example. One could expect the output "Cat Dog Rabbit ". The actual output however is "undefinedCat Dog Rabbit ". In the first iteration of the for-loop, we try to concatenate a string to the value of outputString, which is undefined. JavaScript now performs a implicit coercion and converts the value undefined to the string "undefined".

```
var pets = ["Cat", "Dog", "Rabbit"];

var outputString;

for (var i in pets) {
    outputString += pets[i] + " ";
}

console.log(outputString);
```

Listing 1.1. Implicit Coercions

As we saw, dynamic languages do not require programmers to annotate their programs with type information or to follow any strict typing discipline. This freedom allows programmers to write concise code in short time. However, most code does follow implicit type rules, e.g. only a single type per variable or object property or fixed function signatures. The authors of TypeDevil also state that many bugs are actually violations of these rules. So the freedom offered by dynamic languages often comes at the cost of hidden bugs. Since the language does not enforce any typing discipline, no compile-time warnings are reported if a program uses and combines types inconsistently. Although the code example seems trivial, when you consider critical applications like authentication or payment libraries, of course even a little type error could do some real damage.

3 TypeDevils Approach

3.1 Overview

3.2 Gathering Type Observations

3.3 Building the Type Graph and Identifying Inconsistent Types

3.4 Merging and Pruning of Warnings

The online version of the volume will be available in LNCS Online. Members of institutes subscribing to the Lecture Notes in Computer Science series have access to all the pdfs of all the online publications. Non-subscribers can only read as far as the abstracts. If they try to go beyond this point, they are automatically asked, whether they would like to order the pdf, and are given instructions as to how to do so.

Please note that, if your email address is given in your paper, it will also be included in the meta data of the online version.

4 Related Research

The correct BibTeX entries for the Lecture Notes in Computer Science volumes can be found at the following Website shortly after the publication of the book: `http://www.informatik.uni-trier.de/~ley/db/journals/lncs.html`

Acknowledgments. The heading should be treated as a subsubsection heading and should not be assigned a number.

5 Static Type Systems as an Alternative to Dynamic Analysis

6 Evaluation

6.1 Original Results for TypeDevil

6.2 Comparison between TypeDevil and TypeScript

Introduction to TypeScript

Evaluation Setup

Results

6.3 Discussion

In order to permit cross referencing within LNCS-Online, and eventually between different publishers and their online databases, LNCS will, from now on, be standardizing the format of the references. This new feature will increase the visibility of publications and facilitate academic research considerably. Please base your references on the examples below. References that don't adhere to this style will be reformatted by Springer. You should therefore check your references thoroughly when you receive the final pdf of your paper. The reference section must be complete. You may not omit references. Instructions as to where to find a fuller version of the references are not permissible.

We only accept references written using the latin alphabet. If the title of the book you are referring to is in Russian or Chinese, then please write (in Russian) or (in Chinese) at the end of the transcript or translation of the title.

Please note that proceedings published in LNCS are not cited with their full titles, but with their acronyms!

References

1. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
2. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)

4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov`

Appendix: Springer-Author Discount

LNCS authors are entitled to a 33.3% discount off all Springer publications. Before placing an order, the author should send an email, giving full details of his or her Springer publication, to `orders-HD-individuals@springer.com` to obtain a so-called token. This token is a number, which must be entered when placing an order via the Internet, in order to obtain the discount.

7 Conclusion and Future Work

Here is a checklist of everything the volume editor requires from you:

☐ The final LaTeX source files

☐ A final PDF file

☐ A copyright form, signed by one author on behalf of all of the authors of the paper.

☐ A readme giving the name and email address of the corresponding author.