JavaScript API Misuse Detection by Using TypeScript

[Extended Abstract]

Jihyeok Park
KAIST
jhpark0223@kaist.ac.kr

Abstract

Static analysis of JavaScript programs to detect errors in them is a challenging task. Especially when the program imports massive JavaScript libraries such as jQuery and MooTools, analyzing the whole program and the libraries is expensive and extremely declines the analysis efficiency. In this paper, we introduce a novel approach to solve the problem by modularizing the analysis. We separate the analysis of JavaScript libraries by using types extracted from their corresponding specifications in TypeScript and the analysis of JavaScript applications by a static analysis framework. We use DefinitelyTyped, an open-source repository that provides TypeScript declaration files of over 300 popular JavaScript libraries, and we extend SAFE, an open-source analysis framework for JavaScript.

1. Introduction

JavaScript [1] is the most prevalent language for client-side scripting of web pages. It allows developers to build dynamic web pages by interacting with users and altering document contents in real time. Thanks to these features, 98 out of the 100 most visited websites according to Alexa [5] use JavaScript for client-side programming. Furthermore, the range of JavaScript applications extends to outside of client-side programming for web pages. For example, node.js enables server-side programming in JavaScript to build general scalable network applications for example, and Samsung Smart TV SDK provides support for TV applications in JavaScript. Because of the wide usage of JavaScript, error detection in JavaScript applications has become an important problem recently.

Analysis of JavaScript applications is more challenging than analysis of programs in other programming language such as C or Java because JavaScript does not have a static type system. Although JavaScript is an object-oriented language, objects instead of classes are bases of the JavaScript object-oriented system. Thus, objects are able to inherit properties from others or to add and delete some methods. Moreover, the types of object members are not fixed. Such flexible type structures of JavaScript hinder error detection of JavaScript applications.

We develop a new analysis to detect misuses of JavaScript APIs in JavaScript applications by using their specifications written as TypeScript [12] types in a modular manner. To support various JavaScript APIs in TypeScript, a GitHub repository, DefinitelyTyped [1], provides TypeScript type definitions for over 300 frequently used JavaScript libraries. We build a tool to parse and analyze these specifications to check whether JavaScript programs use the libraries as specified by their corresponding specifications.

2. Misuse Detection by TypeScript

We take account of common cases of JavaScript API misuses in JavaScript applications:

1. Accesses to absent objects or functions in APIs
2. Accesses to absent member properties of API objects
3. Wrong number of arguments to function calls
4. Wrong types of arguments to function calls

Microsoft has developed TypeScript, an open-source programming language that is a strict superset of JavaScript with static typing and class-based object-oriented programming. Thus, any existing JavaScript programs are also valid TypeScript programs. Using TypeScript, static typing via type annotations enables type checking at compile time. TypeScript provides both declarations with the extension `.d.ts` and implementations with the extension `.ts`, where the former include interfaces and the latter include concrete code. The declaration files assign types to API objects with newly defined types as `interface`, `class`, and `enum`. To allow TypeScript programs to use existing JavaScript libraries, the DefinitelyTyped project provides a set of TypeScript declaration files for frequently used JavaScript libraries.

We develop a tool to detect misuses of JavaScript APIs in JavaScript applications by using TypeScript declaration files of JavaScript libraries as semantic criteria. We build the tool as an extension of SAFE [9, 10], a scalable analysis framework for JavaScript. Therefore, we focus on extending SAFE with type definitions for JavaScript APIs. Because the default static analyzer in SAFE is not yet scalable to analyze huge JavaScript libraries such as jQuery and MooTools, we use TypeScript specifications of such libraries as modeling of them for the static analyzer in SAFE.

Figure 1 describes the overall structure of our tool. The shaded box denotes a simplified structure of SAFE; solid boxes denote data and dashed boxes denote modules that transform data. SAFE takes a JavaScript web application; PreProcessor parses the application into JavaScript files and HTML files; HTML Parser generates DOM Trees from HTML files; Parser parses JavaScript programs and translates them into Abstract Syntax Trees (ASTs). We extend SAFE first with TypeScriptRewriter, which substitutes AST nodes relevant to libraries into different AST nodes specially

[1] `https://github.com/borisyankov/DefinitelyTyped`

MODULARITY '14, April 22–26, 2014, Lugano, Switzerland.
Copyright is held by the owner/author(s).
ACM 978-1-4503-2773-2/14/04.
http://dx.doi.org/10.1145/2584469.2584472

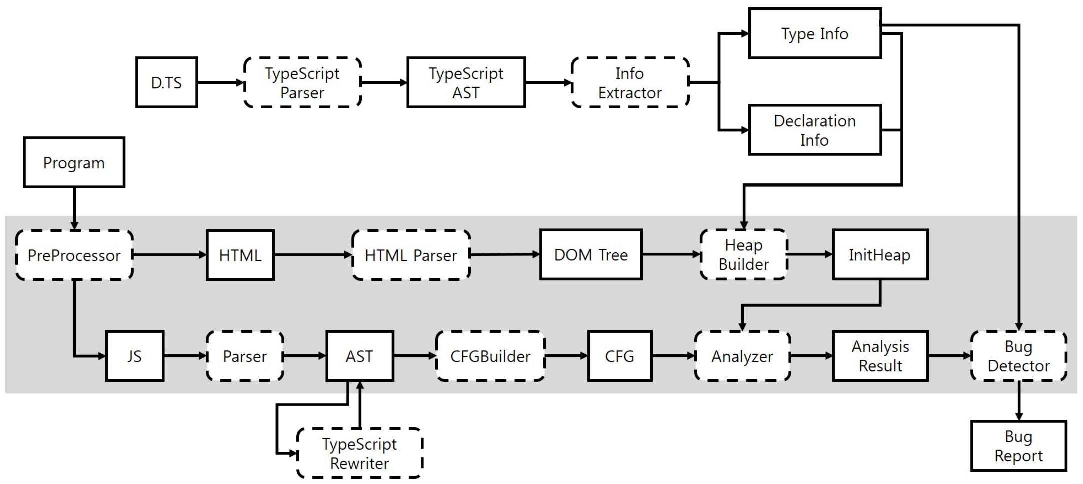

Figure 1. Flow graph of SAFE with TypeScript

marked for later analysis. The main component, Analyzer, receives a CFG and InitHeap including the initial information for analysis of any applications, and analyzes the target JavaScript application. Heap Builder in SAFE builds InitHeap from a DOM Tree and we extend it to take the TypeScript declarations into account. TypeScript Parser parses TypeScript declaration files into TypeScript ASTs and stores them as database files for later uses by the analyzer. Info Extractor builds type information and declaration information from the database files and Heap Builder uses them to build InitHeap. Finally, Bug Detector reports misuses of libraries in the JavaScript application using the analysis results.

Our tool checks whether a JavaScript program correctly uses elements of JavaScript libraries as described by their corresponding specifications in TypeScript. TypeScript declaration files consist of definitions of new types and declarations of variables. Our tool utilizes such information to detect misuses of library elements. For example, a declaration file of jQuery defines `'$'` as `JQueryStatic`, an interface representing static members of jQuery in the same file:

```
interface JQueryStatic { ...
    get(url: string,
        success?: (data: any, textStatus: string,
                   jqXHR: JQueryXHR) => any,
        dataType?: string): JQueryXHR;
... }
declare var $: JQueryStatic;
```

When a JavaScript code calls the `get` function as follows:

```
var x = 'test', y = 42;
$.get(x, y);
```

our tool checks the call using its declaration, which can take 1 to 3 arguments. The first and the third arguments should have the `string` type, and the second argument should have a function. The SAFE Analyzer analyzes that `x` has the `string` type and `y` has the `number` type. Therefore, our tool reports a bug that the second argument does not have a function type. If the code passes the arguments checking, the return type of the function is analyzed to have the type interface `JQueryXHR`. This analysis is simpler and faster than analyzing the library itself as in the original SAFE analysis.

Researchers have studied syntactic checks [4], type systems [2, 7], static analyses [6, 8], and hybrids of static and dynamic analyses [3, 11] of JavaScript programs. Unfortunately, syntactic checks detect only simple errors, type systems are too strict to develop JavaScript programs freely, and static and dynamic analyses cannot analyze massive JavaScript APIs. However, our tool can check not only syntactic errors, but also semantic errors. It can verify misuses of JavaScript APIs without analyzing such APIs themselves.

3. Conclusion

We propose a new approach to detect misuses of JavaScript libraries by using two separate tools, DefinitelyTyped and SAFE modularly. We build a tool to parse and analyze TypeScript declarations of JavaScript libraries from DefinitelyTyped to store in database files. We extend SAFE to recognize the library type information from the pre-analyzed TypeScript declarations in the database files and add them into the initial information for JavaScript program analysis. Our tool traverses JavaScript programs to find out library API uses to check whether they use the APIs correctly, and reports bugs for misuses of JavaScript libraries.

Acknowledgments

This work is supported in part by the National Research Foundation of Korea (NRF) (Grants 2012-0000469 and 2012-0005256), Microsoft Research Asia, Samsung Electronics, S-Core, and Google.

References

[1] ECMAScript Language Specification. Edition 5.1. `http://www.ecma-international.org/publications/standards/Ecma-262.htm`.

[2] Christopher Anderson, Paola Giannini, and Sophia Drossopoulou. Towards type inference for JavaScript. In *ECOOP 2005*.

[3] Ravi Chugh, Jeffrey A. Meister, Ranjit Jhala, and Sorin Lerner. Staged information flow for JavaScript. In *PLDI 2009*.

[4] Douglas Crockford. JSLint. `http://www.jslint.com`.

[5] Salvatore Guarnieri, Marco Pistoia, Omer Tripp, Julian Dolbyand Stephen Teilhet, and Ryan Berg. Saving the world wide web from vulnerable JavaScript. In *ISSTA 2011*.

[6] Arjun Guha, Shriram Krishnamurthi, and Trevor Jim. Using static analysis for Ajax intrusion detection. In *WWW 2009*.

[7] Phillip Heidegger and Peter Thiemann. Recency types for analyzing scripting languages. In *ECOOP 2010*.

[8] Simon Holm Jensen, Anders Møller, and Peter Thiemann. Type analysis for JavaScript. In *SAS 2009*.

[9] PLRG @ KAIST. SAFE: Scalable Analysis Framework for ECMAScript. `http://safe.kaist.ac.kr`.

[10] Hongki Lee, Sooncheol Won, Joonho Jin, Junhee Cho, and Sukyoung Ryu. SAFE: Formal specification and implementation of a scalable analysis framework for ECMAScript. In *FOOL 2012*.

[11] Sergio Maffeis, John C. Mitchell, and Ankur Taly. Isolating JavaScript with filters, rewriting, and wrappers. In *ESORICS 2009*.

[12] Microsoft. TypeScript. `http://www.typescriptlang.org`.