Finding Bugs in JavaScript Engines with Differential Testing

Marcelo d'Amorim and Igor Simões (M.Sc. student)
Postal Address: Av. Jornalista Anibal Fernandes, S/N. Cidade Universitária, PE, Brazil, 50.740-560
Email addresses: {damorim, isol2}@cin.ufpe.br
Phones: +55 (81) 2126-8430, ext. 4379 [work], +55 (81) 98800-2010 [mobile]
Affiliation: Federal University of Pernambuco, Department of Computer Science
Contacts: Martin Barbella (mbarbella@google.com), Vilas Jagannath (vilasshekharbj@gmail.com)

Abstract

JavaScript (JS) is a fundamental programming language for the web. Finding bugs in JavaScript engines is an important problem. This proposal describes our differential testing infrastructure for testing JS engines, our current results, and upcoming features to improve bug-finding in JS engines.

1 Problem Space

JavaScript (JS) is one of the most popular programming languages of today [14,22] largely because of its widespread use on the web. As common in software, the JS specification changes regularly[1] to accommodate the pressing demands of the community. These changes often entail sensible changes in engine implementations [15] that could lead to errors, including regressions. Automated test generation can help reduce the cost of testing, but the lack of executable specifications is an important obstacle. In particular, it is an obstacle to find functional errors.

The goal of this project is to find bugs in implementations of JavaScript engines by leveraging diversity across implementations.

Differential testing [16] has been applied in a variety of contexts [8–11, 20, 21, 23] to mitigate the oracle problem, but it has not been thoroughly explored to find functional bugs in JS engines. The closest work was done by Patra and Pradel [19], where they evaluated their proposed language-agnostic fuzzer to find cross-browser HTML+JS discrepancies. This project aims at building and evaluating an infrastructure for differential testing of runtime engines, such as the JS engine or WebAssembly's. The sensible parts of the infrastructure are the checks of input validity (as to reduce waste/cost) and output correctness (as to reduce false positives).

2 Design Space

Fuzzing is a popular technique to generate new test inputs from existing inputs [17] with the typical goal of finding crashes. Several fuzzing methods have been proposed in the past, varying with respect to how new inputs are generated (e.g., coverage-based [1,3], grammar-based [13,18], and random-based [12]). Figure 1 shows a generic pipeline of a fuzzer infrastructure, instantiated to our problem. Arrows indicate data flow; lines with no arrow denote encapsulation. In our case, JS files are provided as seeds to bootstrap the process. These files can originate from the test suites of the engine codebases or from the bug reports in their issue trackers. The fuzzing infrastructure selects randomly one of the inputs to fuzz with a given tool, say the black-box fuzzer Radamsa [12]. Each fuzzing iteration produces a new file. For grammar-agnostic fuzzers, like Radamsa, the infrastructure can discard syntactically invalid inputs using an external parser for the input language; semantically invalid inputs can be prevented with with linters. Once the input is found to be likely valid, the infrastructure invokes the oracle to check correctness. In case all engines consider the test output consistent (i.e., all pass or all fail), the infrastructure discards the corresponding input. Otherwise, it considers that input as potentially fault-revealing and interesting. Currently, we classify alarms in the HI and LO groups. The HI group includes those warnings that manifest execution errors spawned directly by the test (or its close neighborhood) whereas the LO group includes the rest of the warnings. We found that 88% of the issues reported are from the HI group.

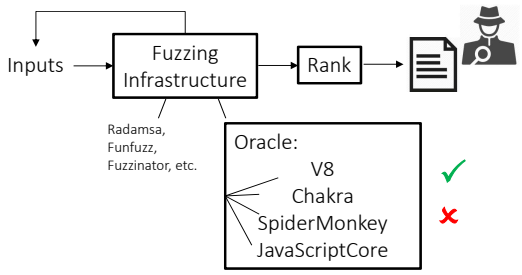

Figure 1: Fuzzing Infrastructure.

[1] Recently, the specification went through a bigger change compared to prior releases, called by the name of ECMAScript6 (ES6) [2]

3 Current Results

The workflow described in Section 2 is currently functional. This section shows preliminary results obtained with our infrastructure providing some evidence that finding bugs in JS runtime engines with differential testing is promising. Table 1 shows the list of bugs we reported on issue trackers of different engines in the period of 42 days. For example, bug # 6 was manifested by the JS code `var a = {valueOf: function(){return ''\x00''}} assert(+a === 0)}`. The object property `valueOf` stores a function that returns a primitive value identifying the target object [5]. That function returns an empty string in the original version of this code. The modified version of the code returns a string representation of a null character (called `NUL` in ascii). The unary plus expression "`+a`", appearing in the assertion, is equivalent to the abstract operation `ToNumber(a.valueOf())` that converts a string to a number, otherwise the operation returns NaN (Not a Number) [6]. For this case, Chakra evaluates the unary plus to NaN as expected, as the null character cannot be converted. As result, the test fails as expected. Chakra, in contrast, incorrectly converts the string to zero, making the test to pass.

Table 1: List of bug reports issued by our team from April 12 to May 24, 2018.

Issue#	Date	Fuzz	Engine	Status	Url
1	4/12	Y	Chakra	**Fixed**	#4978
2	4/12	Y	Chakra	Rejected	#4979
3	4/14	Y	JavascriptCore	New	#184629
4	4/18	N	JavascriptCore	New	#184749
5	4/23	N	Chakra	**Confirmed**	#5033
6	4/25	Y	Chakra	**Fixed**	#5038
7	4/29	N	Chakra	**Confirmed**	#5065
8	4/29	N	Chakra	**Confirmed**	#5067
			JavascriptCore	New	#185130
9	4/29	Y	JavascriptCore	New	#185127
10	4/30	Y	Chakra	**Confirmed**	#5076
			JavascriptCore	New	#185156
11	5/02	Y	JavascriptCore	New	#185197
12	5/02	N	JavascriptCore	New	#185208
13	5/10	Y	Chakra	**Confirmed**	#5128
14	5/17	Y	Chakra	**Confirmed**	#5182
15	5/17	N	Chakra	**Confirmed**	#5187
16	5/21	N	Chakra	**Confirmed**	#5203
17	5/24	Y	JavascriptCore	New	#185943

So far, ten of the bugs we reported were confirmed, two of which were fixed. One bug report we submitted was rejected on the basis that the offending JS file manifested an expected incompatibility across engine implementations. Note from the table that all bug reports still waiting for confirmation are associated with the JavaScriptCore engine (JSC). A closer look at the JSC issue tracker showed that the triage process is very slow for that engine. As of now, we did not find any new bug on SpiderMonkey and V8; the bugs we found were duplicates and were not reported. Finally, it is worth noting that 7 of the 17 JS files that manifested discrepancies were *not* produced with fuzzing (column "Fuzz"). These are test files from suites of different engines. This observation emphasizes the importance of continuously collecting test suites from multiple sources; today, we use test suites from seven different open source engines, including a total of 30K test files.

4 Future Steps

The list below shows future tasks for this project.

- Integrate different fuzzers, including AFL [1], HonggFuzz [3], LibFuzzer [4], jsfunfuzz [18], and Grammarinator [13].
- Find balance between generational strategies (, which often produce valid but simple inputs) and mutational strategies (, which often produce complex but invalid inputs) for fuzzing.
- Explore different strategies to reduce the number of false positives as to facilitate the inspection process. For example, mining issue trackers for similar bug reports is a candidate alternative.
- Look for bugs in WebAssembly [7], a new binary format already supported by major browsers. In principle, our infrastructure is agnostic to the runtime engine used. We are eager to explore other sources of bugs, especially in very new engines.

5 Data Policy

The results produced with this research will be made available to the public and to the research community. All tools and data sets developed will be available online, and the software will be released under an open source license.

References

[1] American fuzz loop. http://lcamtuf.coredump.cx/afl/.

[2] Ecmascript6 new features. http://es6-features.org.

[3] Honggfuzz. https://github.com/google/honggfuzz.

[4] Libfuzzer. https://llvm.org/docs/LibFuzzer.html.

[5] Object.valueof documentation. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/ValueOf.

[6] Unary plus - es6 specifications. https://www.ecma-international.org/ecma-262/8.0/index.html#sec-unary-plus-operator.

[7] Webassembly. https://webassembly.org/.

[8] George Argyros, Ioannis Stais, Suman Jana, Angelos D. Keromytis, and Aggelos Kiayias. Sfadiff: Automated evasion attacks and fingerprinting using black-box differential automata learning. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1690–1701, New York, NY, USA, 2016. ACM.

[9] David Brumley, Juan Caballero, Zhenkai Liang, James Newsome, and Dawn Song. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 15:1–15:16, Berkeley, CA, USA, 2007. USENIX Association.

[10] Yuting Chen, Ting Su, Chengnian Sun, Zhendong Su, and Jianjun Zhao. Coverage-directed differential testing of jvm implementations. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, pages 85–99, New York, NY, USA, 2016. ACM.

[11] Yuting Chen and Zhendong Su. Guided differential testing of certificate validation in ssl/tls implementations. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 793–804, New York, NY, USA, 2015. ACM.

[12] Aki Helin. Radamsa fuzzer. https://github.com/aoh/radamsa.

[13] Renata Hodovan. Grammarinator. https://github.com/renatahodovan/grammarinator.

[14] Business Insider. The 15 most popular programming languages, according to the 'facebook for programmers'. http://www.businessinsider.com/the-9-most-popular-programming-languages-according-to-the-facebook-for-programmers-2017-10.

[15] Kangax. Ecmascript6 compatibility. http://kangax.github.io/compat-table/es6/.

[16] William M. McKeeman. Differential testing for software. *DIGITAL TECHNICAL JOURNAL*, 10(1):100–107, 1998.

[17] Barton P. Miller. Fuzz testing. http://pages.cs.wisc.edu/~bart/fuzz/.

[18] Mozilla. jsfunfuzz. https://github.com/MozillaSecurity/funfuzz/tree/master/src/funfuzz/js/jsfunfuzz.

[19] Jibesh Patra and Michael Pradel. Learning to fuzz: Application-independent fuzz testing with probabilistic, generative models of input data. Technical report, TU Darmstadt, Department of Computer Science, Nov 2016.

[20] T. Petsios, A. Tang, S. Stolfo, A. D. Keromytis, and S. Jana. Nezha: Efficient domain-independent differential testing. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 615–632, May 2017.

[21] Suphannee Sivakorn, George Argyros, Kexin Pei, Angelos D. Keromytis, and Suman Jana. Hvlearn: Automated black-box analysis of hostname verification in SSL/TLS implementations. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 521–538, 2017.

[22] Stackify. Most popular and influential programming languages of 2018. https://stackify.com/popular-programming-languages-2018/.

[23] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in c compilers. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 283–294, New York, NY, USA, 2011. ACM.

Marcelo d'Amorim

Associate Professor, Computer Science Department
Federal University of Pernambuco (UFPE), Recife, Brazil

Address
Universidade Federal de Pernambuco - UFPE - CIn
Av. Jornalista Anibal Fernandes, S/N
Cidade Universitária, PE, Brazil 50.740-560

Email ..... damorim@cin.ufpe.br
Personal Page ..... http://www.cin.ufpe.br/~damorim/
Skype id ..... mdamorim
Work Phone ..... +55 (81) 2126-8430 ext. 4379

Research Interests My research interests are in the areas of Software Engineering and Programming Languages, with a focus on improving software reliability through program analysis and systematic testing.

Professional Preparation

University of Illinois at Urbana-Champaign ..... Illinois, USA
2007 Ph.D. in Computer Science
Dissertation title: "Efficient Explicit-State Model Checking for Programs with Dynamically Allocated Data"
Advisor: Prof. Darko Marinov

Federal University of Pernambuco ..... Pernambuco, Brazil
2001 M.S. in Computer Science
1997 B.S. in Computer Science

Appointments

01/09– **Federal University of Pernambuco** ..... Pernambuco, Brazil
Associate Professor. Advising: 1PhD+2MS+3UROP students, Graduated: 1PhD+3MS, Co-advised: 3MS.

7/15–6/16 **Georgia Institute of Technnology** ..... Atlanta, USA
Visiting Scholar at the Arktos group led by Alessandro Orso

09/07–12/08 **Federal University of Pernambuco** ..... Pernambuco, Brazil
Postdoctoral researcher at the SPG group led by Paulo Borba

05/04–08/04 **NASA Ames Research Center** ..... California, USA
Summer intern. Supervisor: Klaus Havelund

08/02–08/07 **University of Illinois at Urbana-Champaign** ..... Illinois, USA
Research assistant. Advisor: Darko Marinov

Selected Awards & Honors

2016 IEEE/ACM ASE'16 Distinguished Reviewer Award.
2016 CNPq sabbatical scholarship.
2014 Finalist (with 5 others) Microsoft Research Faculty Fellowship Latin America.
2013 Microsoft Software Engineering Innovation Foundation (SEIF) Award 2013.
2013 CNPq research productivity fellowship (current).
2008 FACEPE/CNPq postdoctoral scholarship.
2002 CAPES Ph.D. fellowship (2002-2006).

Publications (2008+)

- M. Fazzini, M. Prammer, M. d'Amorim, and A. Orso. Automatically translating bug reports into test cases for mobile apps. In *International Symposium on Software Testing and Analysis (ISSTA), Amsterdam, Netherlands, July 2018*, 2018. To Appear
- X. Li, S. Zhu, M. d'Amorim, and A. Orso. Enlightened debugging. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 2018.*, 2018. To Appear
- S. Souto and M. d'Amorim. Time-space efficient regression testing for configurable systems. *Journal of Systems and Software*, 137:733–746, 2018
- S. Souto, M. d'Amorim, and R. Gheyi. Balancing soundness and efficiency for practical testing of configurable systems. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 632–642, 2017
- J. Candido, L. Melo, and M. d'Amorim. Test suite parallelization in open-source projects: a study on its usage and impact. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 838–848, 2017
- X. Li, M. d'Amorim, and A. Orso. Iterative user-driven fault localization. In *Hardware and Software: Verification and Testing - 12th International Haifa Verification Conference, HVC 2016, Haifa, Israel, November 14-17, 2016, Proceedings*, pages 82–98, 2016

- M. Borges, A. Filieri, M. d'Amorim, and C. S. Păsăreanu. Iterative distribution-aware sampling for probabilistic symbolic execution. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE, pages 866–877, New York, NY, USA, 2015. ACM
- P. Barros, R. Just, S. Millstein, P. Vines, W. Dietl, M. d'Amorim, and M. D. Ernst. Static Analysis of Implicit Control Flow: Resolving Java Reflection and Android Intents. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE, pages 669–679, Washington, DC, USA, 2015. IEEE Computer Society
- S. Souto, D. Gopinath, M. d'Amorim, D. Marinov, S. Khurshid, and D. Batory. Faster bug detection for software product lines with incomplete feature models. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC, pages 151–160, New York, NY, USA, 2015. ACM
- A. Rimsa, M. d'Amorim, F. M. Q. a. Pereira, and R. S. Bigonha. Efficient static checker for tainted variable attacks. *Science of Computer Programming*, 80:91–105, Feb. 2014
- M. Borges, A. Filieri, M. d'Amorim, C. S. Păsăreanu, and W. Visser. Compositional solution space quantification for probabilistic software analysis. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI, pages 123–132, New York, NY, USA, 2014. ACM
- Q.-S. Phan, P. Malacaria, C. S. Păsăreanu, and M. d'Amorim. Quantifying information leaks using reliability analysis. In *Proceedings of the International SPIN Symposium on Model Checking of Software*, SPIN, pages 105–108, New York, NY, USA, 2014. ACM
- T. Liu, M. Araújo, M. d'Amorim, and M. Taghdiri. A comparative study of incremental constraint solving approaches in symbolic execution. In *Proceedings of the 10th International Haifa Verification Conference*, HVC, pages 284–299, 2014
- C. H. P. Kim, D. Marinov, S. Khurshid, D. Batory, S. Souto, P. Barros, and M. d'Amorim. SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE, pages 257–267, New York, NY, USA, 2013. ACM
- J. Campos, R. Abreu, G. Fraser, and M. d'Amorim. Entropy-based test generation for improved fault localization. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ASE, pages 257–267, Washington, DC, USA, 2013. IEEE Computer Society
- M. Borges, M. d'Amorim, S. Anand, D. H. Bushnell, and C. S. Pasareanu. Symbolic execution with interval solving and meta-heuristic search. In *Proceedings of the Fifth IEEE International Conference on Software Testing, Verification and Validation*, ICST, pages 111–120, 2012
- M. Souza, M. Borges, M. d'Amorim, and C. S. Pasareanu. CORAL: solving complex constraints for symbolic pathfinder. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, pages 359–374, 2011
- A. Rimsa, M. d'Amorim, and F. M. Q. a. Pereira. Tainted flow analysis on e-ssa-form programs. In *Proceedings of the 20th International Conference on Compiler Construction: Part of the Joint European Conferences on Theory and Practice of Software*, CC/ETAPS, pages 124–143, Berlin, Heidelberg, 2011. Springer-Verlag
- E. Alves, M. Gligoric, V. Jagannath, and M. d'Amorim. Fault-localization using dynamic slicing and change impact analysis. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, pages 520–523, 2011
- M. Takaki, D. Cavalcanti, R. Gheyi, J. Iyoda, M. d'Amorim, and R. B. C. Prudêncio. Randomized constraint solvers: a comparative study. *ISSE*, 6(3):243–253, 2010,
- A. Sobeih, M. d'Amorim, M. Viswanathan, D. Marinov, and J. C. Hou. Assertion checking in j-sim simulation models of network protocols. *Simulation*, 86(11):651–673, 2010
- C. Bertolini, G. Peres, M. d'Amorim, and A. Mota. An empirical evaluation of automated black box testing techniques for crashing guis. In *Second International Conference on Software Testing Verification and Validation, ICST 2009, Denver, Colorado, USA, April 1-4, 2009*, pages 21–30, 2009
- M. Takaki, D. Cavalcanti, R. Gheyi, J. Iyoda, M. d'Amorim, and R. B. C. Prudêncio. A comparative study of randomized constraint solvers for random-symbolic testing. In *First NASA Formal Methods Symposium - NFM 2009, Moffett Field, California, USA, April 6-8, 2009.*, pages 56–65, 2009
- T. Gvero, M. Gligoric, S. Lauterburg, M. d'Amorim, D. Marinov, and S. Khurshid. State extensions for java pathfinder. In *30th Intl. Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 863–866, 2008
- M. d'Amorim, S. Lauterburg, and D. Marinov. Delta execution for efficient state-space exploration of object-oriented programs. *IEEE Transactions on Software Engineering*, 34(5):597–613, 2008