Montage: A Neural Network Language Model-Guided JavaScript Engine Fuzzer

Suyoung Lee, HyungSeok Han, Sang Kil Cha, Sooel Son

KAIST
Popularity of Web Browsers

4 billion users

JS Engine Vulnerabilities

Costs up to $500,000

Source: https://zerodium.com/program.html
How can a fuzzer generate JS inputs?
Previous Work

1. Mutation-based fuzzers
   - LangFuzz, IFuzzer, and GramFuzz
   - Combining AST subtrees extracted from JS seeds

2. Generation-based fuzzers
   - jsfunfuzz
   - Applying JS grammar rules from scratch
Previous Work – Building Blocks

1. Mutation-based fuzzers
   - LangFuzz, IFuzzer, and GramFuzz
   - Combining AST subtrees extracted from JS seeds

2. Generation-based fuzzers
   - jsfunfuzz
   - Applying JS grammar rules from scratch
They randomly select building blocks!
Patterns of Bug-triggering Code

```
let v0 = new Array(0x10000);
v0 = v0.fill(0x1234).join(', ');
eval('new Array(' + v0 + ')');
```

Bug-triggering JS code

Extracted Patterns
Patterns of Bug-triggering Code

Current AST

Extracted Building Blocks

Which building block is more likely to trigger JS engine bugs?
Study on JS Engine Vulnerabilities

1. Functional commonalities

2. Syntactical commonalities
Functional Commonalities

• Study 1. Patches of 50 ChakraCore CVEs

| CVE-2017-0071 | 18% revised GlobOpt.cpp |
| CVE-2017-0141 | 18% are related to global optimization |
| CVE-2017-0196 | 14% revised JavascriptArray.cpp |
| : | = | 14% are related to JavaScript arrays |
| CVE-2018-0953 | |

Patches
Syntactical Commonalities

- Study 2. AST subtrees from two sets

At August, 2016

Extracted subtrees

After August, 2016

Extracted subtrees

95% already existed!

2,038 JS tests from ChakraCore repo

CVE-2018-0980

67 PoCs triggering ChakraCore CVEs

2,038 JS tests from ChakraCore repo
Syntactical Commonalities

• Study 2. AST subtrees from two sets

```javascript
function f0() {
  'use asm';
  const v0 = Math.fround(1);
  function f1() {
    var v1 = v0;
    var v2 = Math.fround(4);
  }
  return {f1: f1}
}

JS test from ChakraCore repo
Syntactical Commonalities

• Study 2. AST subtrees from two sets

```javascript
function f0() {
    'use asm';
    const v0 = 1.0;
    function f1() {
        var v1 = v0;
        var v0 = 0;
    }
    return f1;
}
```

PoC (CVE-2017-11911)
Our Approach

1. Functional commonalities
   - Mutating existing JS regression tests

2. Syntactical commonalities
   - Modeling of the relationships between AST subtrees
Montage Overview

1. **Preprocessing**
   - A sequence of fragments

2. **Training**
   - A trained model

3. **AST mutation**
   - JS’
**Preprocessing**

```javascript
const v0 = 10;
```

![JavaScript AST Diagram]

*JavaScript AST*
Preprocessing

```javascript
const v0 = 10;
```

<Diagram of JavaScript AST>

**JavaScript AST**

**Pre-order traversal**

A sequence of fragments
Preprocessing

```javascript
const v0 = 10;
```

**JavaScript AST**

**A sequence of fragments**
Preprocessing

```javascript
const v0 = 10;
```

![JavaScript AST](image1)

Pre-order traversal

A sequence of fragments
Preprocessing

const v0 = 10;

JavaScript AST
Preprocessing

```javascript
const v0 = 10;
```

**JavaScript AST**

- Program
  - body
- VarDeclaration
  - kind
  - decl
  - id
  - init
- Identifier
  - name
  - v0
- Literal
  - value
  - 10

**Pre-order traversal**

A sequence of fragments
const v0 = 10;

JavaScript AST

A Sequence of Fragments
Global Compositional Relationship

```javascript
const v0 = 10;
```

JavaScript AST

A sequence of fragments
vs. A Sequence of Tokens [1-3]

const v0 = 10;

% of Valid JS code

0.58% vs. 58.26%

Token-level model

Fragment-level model

A sequence of tokens

JavaScript AST

Selecting Valid Fragments

const v0 = 10;

JavaScript AST

A sequence of fragments
Applicable to Any Language Models

const v0 = 10;

A sequence of fragments

Language models

Training an LSTM model

A sequence of preceding fragments → LSTM model → Probability distribution of a next fragment
var v0 = 'Hello World';
v1 = [];
f0();

function f0 () {
    v1.valueOf();
    var v2 = 10;
}
AST Mutation

```javascript
var v0 = 'Hello World';
v1 = [];
function f0 () {
    v1.valueOf();
    var v2 = 10;
}
```
AST Mutation

A sequence of fragments representing the current AST

Trained LSTM model
AST Mutation

The probability distribution of the next fragment:

\[
\begin{bmatrix}
0.76 & 0.12 & 0.001
\end{bmatrix}
\]
AST Mutation

Pre-order traversal

Randomly select one from the Top K fragments

- ExprStament
- CallExpr
  - callee
  - args
- MemberExpr
- Literal

Trained LSTM model

0.76  0.12  0.001
AST Mutation

- ExprStatement
  - body
  - expr
- CallExpr
  - callee
  - args
- MemberExpr
  - object
  - property
- Identifier
- Identifier
- Literal

Trained LSTM model
AST Mutation

Trained LSTM model
AST Mutation

Trained LSTM model
AST Mutation

Trained LSTM model
AST Mutation

var v0 = ‘Hello World’;
v1 = [];
f0();

function f0 () {
    v1.push(0x10);
    var v2 = 10;
}
Experiment Setup

- Collected **33.5K** unique JS files
  - Regression tests from the repo of four major JS engines and Test262
  - PoCs of known CVEs
- Ran fuzzers against **ChakraCore 1.4.1**
- JS code triggering unpatched bugs is not in our training set!
vs. State-of-the-art Fuzzers

- **72 hours x 5 trials**
  - CodeAlchemist: A *state-of-the-art* semantics-aware JS fuzzer, *NDSS’19*
  - jsfunfuzz: A JS fuzzer developed by Mozilla
  - IFuzzer: An evolutionary JS fuzzer, *ESORICS’16*

<table>
<thead>
<tr>
<th>Metric</th>
<th>Build</th>
<th># of Unique Crashes (Known CVEs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Montage</td>
</tr>
<tr>
<td>Median</td>
<td>Release</td>
<td>23 (7)</td>
</tr>
<tr>
<td></td>
<td>Debug</td>
<td>49 (12)</td>
</tr>
</tbody>
</table>

The differences were **statistically significant** ($p$-value < 0.05)!
A Sequence of Fragments vs. Tokens [1-3]

- **72 hours x 5 trials**
- Token RNN: JS code mutation guided by a token-level LSTM model

<table>
<thead>
<tr>
<th>Metric</th>
<th>Build</th>
<th># of Unique Crashes (Known CVEs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Montage</td>
<td>Token RNN</td>
</tr>
<tr>
<td>Median</td>
<td>Release</td>
<td>23 (7)</td>
</tr>
<tr>
<td></td>
<td>Debug</td>
<td>49 (12)</td>
</tr>
</tbody>
</table>

Is the LSTM Model Effective?

• 72 hours x 5 trials
  - Random: Random fragment assembly without any model

<table>
<thead>
<tr>
<th>Metric</th>
<th>Build</th>
<th># of Unique Crashes (Known CVEs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Montage</td>
</tr>
<tr>
<td>Median</td>
<td>Release</td>
<td>23 (7)</td>
</tr>
<tr>
<td></td>
<td>Debug</td>
<td>49 (12)</td>
</tr>
</tbody>
</table>
Is the LSTM Model Effective?

- The # of appended fragments to compose a new subtree

**Captured long-term dependencies!**
Finding Real-World Bugs

• Four major JS engines for a total of 1.5 months
  - Found 37 previously unknown bugs
    ➢ 34 bugs including two CVEs from ChakraCore 1.11.7
    ➢ One bug from V8 7.4.0 (beta)
    ➢ Two bugs including one CVE from JavaScriptCore 2.23.3
  - 26 of them were patched at the time of writing

Microsoft rewarded with $5,000!
Conclusion

• We proposed the first neural network language model-guided JS engine fuzzer and demonstrated its efficacy.

• We proposed a novel approach of modeling JS code as a sequence of fragments on which any prevalent language models can be trained without modification.

• Montage outperformed state-of-the-art fuzzers in the old version of ChakraCore.

• Montage found 37 previously unreported bugs from the latest JS engines.
Open Science

https://github.com/WSP-LAB/Montage
For More Details

- Resolving reference errors
- Effect of parameters
- Effect of language models
- Case studies
Question?

If you have more questions, please email suyoung.lee@kaist.ac.kr.