JSTAP: A Static Pre-Filter for Malicious JavaScript Detection

ACSAC‘19 | 11-12-19

Aurore Fass, Michael Backes, and Ben Stock

CISPA Helmholtz Center for Information Security
Motivation

Motivation for Js Samples:

- CUJO
- Zozzle
- JAST

Static detector

Benign

Malicious

Dynamic detector

Aurore Fass – JStap
Contributions

- Entirely static analysis with additional semantic information:
  - Control flow: reasons about conditions
  - Data flow: reasons about execution order
  - Added value?

- Are prior works still able to detect current malicious JS?

- Which static code abstraction performs best?
JSTAP Overview

Abstract Code Representations

- Tokens
- AST
- CFG
- PDG
- PDG-DFG

Features Extraction

- ngrams
- value

Select 1

Abstract Code Representation

Select 1

Features Extraction

Select 1

Machine Learning

Predictions

JSTAP module

Aurore Fass – JStap
JSTAP Overview

Abstract Code Representations

- Tokens
- AST
- CFG
- PDG
- PDG-DFG

Features Extraction

- ngrams
- value

Predictions

Select 1

Abstract Code Representation

Select 1

Features Extraction

Select 1

Machine Learning

JSTAP module
Abstract Code Representation: Tokens

1. `x . if = 1 ;`
2. `var y = 1 ;`
3. `if ( x . if == 1 ) { d = y ; }`

➢ **Tokens**: linear conversion into abstract symbols
Abstract Code Representation: Tokens

1. x.if
2. var y = 1;
3. if (x.if == 1) {d = y;}

- Identifier
- Punctuator
- Keyword

- Tokens: linear conversion into abstract symbols
Abstract Code Representation: Tokens

1. \(x\text{.if} = 1;\)
2. \(\text{var } y = 1;\)
3. \(\text{if } (x\text{.if} == 1) \{d = y;\}\)

- Identifier Punctuator Keyword Punctuator Numeric Punctuator
- Keyword Identifier Punctuator Numeric Punctuator
- Keyword Punctuator Identifier Punctuator Keyword Punctuator Numeric Punctuator Punctuator Identifier Punctuator Identifier Punctuator Punctuator

➤ Tokens: linear conversion into abstract symbols
AST (Abstract Syntax Tree): how the programming constructs are nested

1. `x.if = 1;
2. var y = 1;
3. if (x.if == 1) {d = y;}`
Abstract Code Representation: CFG

1. Expression Statement
   - Assignment Expression
     - Member Expression
       - Id
       - Id
     - Literal
       - x
       - if
   - Expression
     - Literal
     - Id
     - Id
     - if
     - 1

2. Variable Declaration
   - Variable Declarator
     - Id
     - Literal
     - y
     - 1

3. If Statement
   - Block Statement
     - Expression Statement
       - Assignment Expression
         - Id
         - Id
         - d
         - y
   - True
   - Expression
     - Member Expression
       - Id
       - Id
     - Literal
       - x
       - if
     - 1

CFG (Control Flow Graph): execution path conditions

1. x.if = 1;
2. var y = 1;
3. if (x.if == 1) {d = y;}
Abstract Code Representation: PDG

PDG (Program Dependency Graph): control flow + data dependencies

1. `x.if = 1;`
2. `var y = 1;`
3. `if (x.if == 1) {d = y;}`
Abstract Code Representation: PDG-DFG

PDG-DFG: data dependencies only

1. \( x \text{.if} = 1; \)
2. \( \text{var } y = 1; \)
3. \( \text{if} (x \text{.if} == 1) \{ d = y; \} \)
Abstract Code Representations: Summary

Abstract Code Representation: Tokens
1. x if = 1;
2. var y = 1;
3. if (x, if == 1) { d = y; }

Tokens: linear conversion into abstract code

Abstract Code Representation: CFG
1. Expression Statement
2. Variable Declaration
3. If Statement
4. Assignment Expression
5. Variable Declarator
6. Binary Expression
7. Block Statement
8. Member Expression
9. Literal
10. Member Declarator

CFG (Control Flow Graph): execution path

Abstract Code Representation: PDG-DFG
1. Expression Statement
2. Variable Declaration
3. If Statement
4. Assignment Expression
5. Variable Declarator
6. Binary Expression
7. Block Statement
8. Member Expression
9. Literal
10. Member Declarator

PDG-DFG: data dependencies only

Abstract Code Representation: AST
1. Expression Statement
2. Variable Declaration
3. Assignment Expression
4. Member Expression
5. Literal

AST (Abstract Syntax Tree): how the programming constructs are nested

Abstract Code Representation: PDG
1. Expression Statement
2. Variable Declaration
3. If Statement
4. Assignment Expression
5. Variable Declarator
6. Binary Expression
7. Block Statement
8. Member Expression
9. Literal

PDG (Program Dependency Graph): control flow + data dependencies
JSTAP Overview

Abstract Code Representations
- Tokens
- CFG
- AST
- PDG
- PDG-DFG

Features Extraction
- ngrams
- value

Select 1

Abstract Code Representation
Select 1

Features Extraction
Select 1

Machine Learning

Predictions
1. Select an abstract code representation

2. Traverse it and extract the corresponding units

3. Combine these units in groups of n = build n-grams

Features Extraction: ngrams

AST:

ExpressionStatement
  AssignmentExpression
    MemberExpression
      Id
      Id
      Literal
      Id
      Id
      Literal

ExpressionStatement
  AssignmentExpression
    MemberExpression
      Id
      Id
      Literal

AssignmentExpression
  MemberExpression
    Id
    Id

MemberExpression
  Id
  Id

ExpressionStatement
  AssignmentExpression
    MemberExpression
      Id
      Id
      Literal

AssignmentExpression
  MemberExpression
    Id
    Id

MemberExpression
  Id
  Id
  Literal

Aurore Fass – JStap
1. Select an abstract code representation

2. Traverse it and extract the corresponding units along with their value

AST:

Expression
Statement

Assignment
Expression

Member
Expression

Literal

(Id, x)
(Id, if)
(Int, 1)
Abstract Code Representations
- Tokens
- AST
- CFG
- PDG
- PDG-DFG

Features Extraction
- ngrams
- value

Select 1
Abstract Code Representation
Select 1
Features Extraction
Select 1
Machine Learning

Predictions

JSTAP module

Aurore Fass – JStap
## Experimental Setup

<table>
<thead>
<tr>
<th>Malicious Source</th>
<th>#JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSI</td>
<td>83,361</td>
</tr>
<tr>
<td>Hynek Petrak</td>
<td>29,558</td>
</tr>
<tr>
<td>Kafeine DNC</td>
<td>12,982</td>
</tr>
<tr>
<td>VirusTotal</td>
<td>3,056</td>
</tr>
<tr>
<td>GeeksOnSecurity</td>
<td>2,491</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>131,448</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benign Source</th>
<th>#JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tranco-10k</td>
<td>122,910</td>
</tr>
<tr>
<td>Microsoft</td>
<td>16,271</td>
</tr>
<tr>
<td>Games</td>
<td>1,992</td>
</tr>
<tr>
<td>Web Frameworks</td>
<td>427</td>
</tr>
<tr>
<td>Atom</td>
<td>168</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>141,768</strong></td>
</tr>
</tbody>
</table>

*Total #JS*: 131,448 *Malicious Source* *Benign Source* 141,768

\[ \times 5 \quad 10,000 \quad \text{Model} \quad 10,000 \]
Detection Performance: ngrams

- Best one: AST with 99.38% correct classifications
Detection Performance: value

Best one: Tokens with 99.44% correct classifications
Related Work Comparison: Cujo

![Bar chart comparing TPR and TNR for Cujo and Tokens + ngrams]

- TPR:
  - Cujo: 0.98
  - Tokens + ngrams: 1.00

- TNR:
  - Cujo: 0.98
  - Tokens + ngrams: 1.00

Aurore Fass – JStap
Related Work Comparison: JAST

![Graph showing accuracy comparison between JaSt and AST + ngrams]

- **TPR**: TPR (True Positive Rate) values for JaSt and AST + ngrams.
  - JaSt: 0.90, 0.92, 0.94, 0.96, 1.00
  - AST + ngrams: 0.94, 0.96, 0.98, 1.00
- **TNR**: TNR (True Negative Rate) values for JaSt and AST + ngrams.
  - JaSt: 0.90, 0.92, 0.94, 0.96
  - AST + ngrams: 0.98, 1.00

Aurore Fass – JStap
Related Work Comparison: Zozzle

<table>
<thead>
<tr>
<th></th>
<th>TPR</th>
<th>TNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zozzle</td>
<td>0.92</td>
<td>0.98</td>
</tr>
<tr>
<td>AST + value</td>
<td>0.94</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Modules Combination

Pre-filter 1

AST + ngrams
Tokens + value
PDG + value
Modules Combination

253,216 Samples

Pre-filter 1
- AST + ngrams
- Tokens + value
- PDG + value

Unanimous voting →

Further analysis

Predictions

Accuracy: 99.73%

92.76%

Aurore Fass – JStap
Modules Combination

253,216 Samples

Pre-filter 1
AST + ngrams
Tokens + value
PDG + value

Further analysis

Unanimous voting ➔
92.76%

Predictions

Accuracy: 99.73%

Pre-filter 2
Tokens + ngrams
AST + ngrams
AST + value

7.24%

Aurore Fass – JStap
Modules Combination

253,216 Samples

Pre-filter 1
AST + ngrams
Tokens + value
PDG + value

Further analysis
Unanimous voting →
Predictions
Accuracy: 92.76%

Pre-filter 2
Tokens + ngrams
AST + ngrams
AST + value

7.24%
Unanimous voting →
Predictions
Accuracy: 6.5%

Dynamic analysis

Unanimous voting →
Predictions
Accuracy: 0.74%

92.76%

99.73% Accuracy:

99% Accuracy:
Conclusion

**JSTAP Overview**

- **Abstract Code Representations**
  - Tokens
  - CFG
  - AST
  - PDG
  - PDG-DFG

- **Features Extraction**
  - ngrams
  - value

**Modules Combination**

- **Pre-filter 1**
  - AST + ngrams
  - Tokens + value
  - PDG + value
  - Pre-filter 2
  - Tokens + ngrams
  - AST + value

**Detection Performance: ngrams**

- Accuracy (
  - TPR or TNR)

---

**Thank you**

---

**JSTAP: A Static Pre-Filter for Malicious JavaScript Detection**

Aurore Fass, Michael Backes, and Ben Stock
CISPA Helmholtz Center for Information Security
{aurore.fass, backes, stock}@cispa.saarland

Aurore54F / JStap

@AuroreFass  @kcotsneb