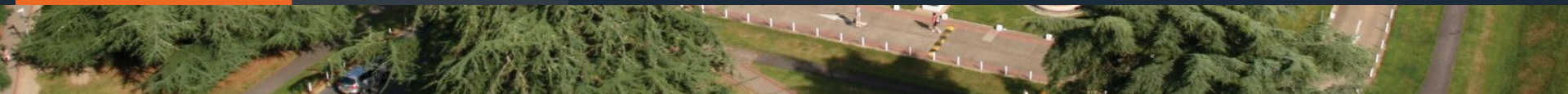

ROYAL HOLLOWAY
UNIVERSITY
OF LONDON

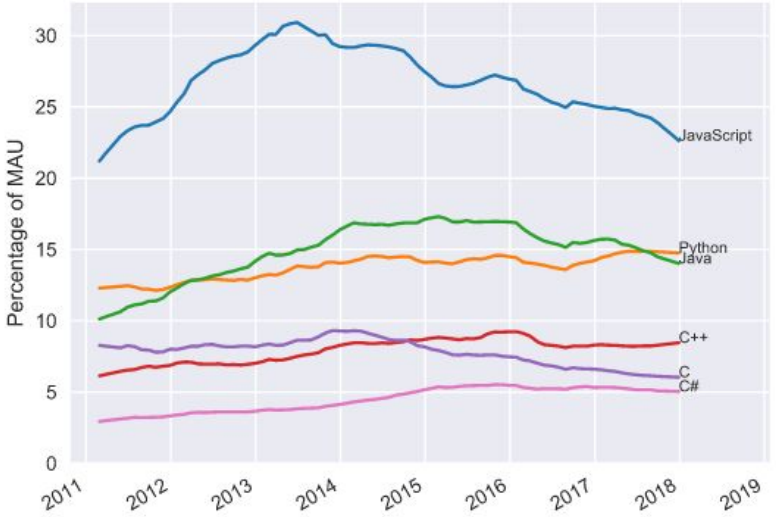
# ExpoSE: Practical Symbolic Execution of JavaScript

**Blake Loring**, Duncan Mitchell, Johannes Kinder



# JavaScript

- The Language of the web.
- Increasingly popular as server-side (Node.js) and client side (Electron) solution.
- Top 10 language (Github)



# JavaScript (2)

- Large & Confusing Specification

```
» typeof null
← "object"
```

- Language constantly revised

| | | | |
|---|---|---|---|
| ECMA-402 2nd Edition | PDF | June 2015 | ECMAScript 2015 Internationalization API Specification |
| ECMA-262 7th Edition | HTML | June 2016 | ECMAScript 2016 Language Specification |
| ECMA-402 3rd Edition | HTML | June 2016 | ECMAScript 2016 Internationalization API Specification |
| ECMA-262 8th Edition | HTML | June 2017 | ECMAScript 2017 Language Specification |
| ECMA-402 4th Edition | HTML | June 2017 | ECMAScript 2017 Internationalization API Specification |

# JavaScript (3)

- JavaScript is currently hard to test / verify
- Static approaches limited by dynamic aspects of the language
- Solutions like Flow / TS their own ecosystem
- Good target for DSE

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

When argument starts with ‘--sanitize’

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

When argument starts with '--sanitize'

When debug coerces to 5 or 'on'

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

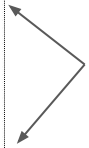
# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```



Heavy use of strings and regular expressions

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

Heavy use of strings and regular expressions

Asynchronous event model

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

Heavy use of strings and regular expressions

Asynchronous event model

Implicit Type Coercion

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

Heavy use of strings and regular expressions

Asynchronous event model

Implicit Type Coercion

Dynamic code generation

# Challenges

```
const match = /(--sanitize )?(.*)/.exec(SYMBOL);
const debug = SYMBOL;

if (match) {
    const [sanitize, url] = match.slice(1);
    let req = requestData(url);

    if (sanitize) {
        req.on('data', x => '' + y);
    }

    req.on('done', result => {
        result = '' + result;
        result = eval(result);
        if (debug == 5 || debug == "on") {
            console.log(urld);
        }
    });
}
```

Heavy use of strings and regular expressions

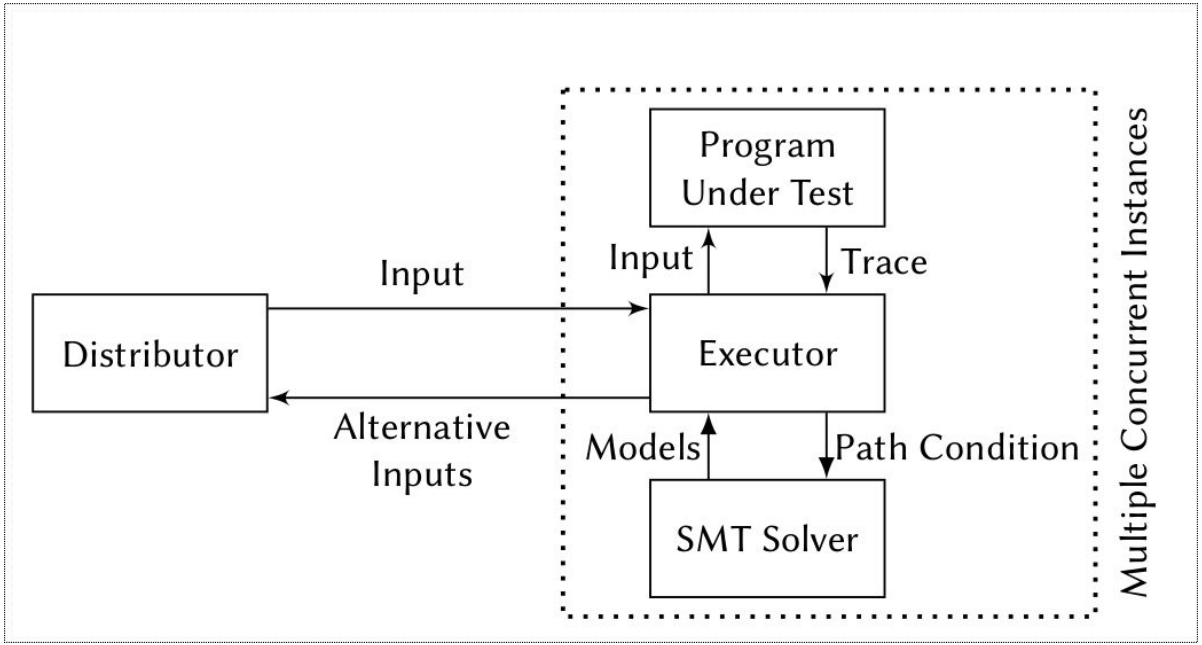
Asynchronous event model

Implicit Type Coercion

Dynamic code generation

**I will address these issues today**

# ExpoSE




**New DSE Tool Open Source**

# Instrumentation

- Instrument program trace into source code

```
if (sanitize) {
    req.on('data', x => '' + y);
}
```

```
if (IfCb(sanitize)) {
    FnCb(GetFieldCb(req, 'on'), data, x => AddCb('', y));
}
```

- Mature instrumentation tools available (Jalangi2, NodeProf, etc)
- Handle eval for free

# Asynchronous Events

```
req.on('done', result => {
     result = '' + result;
     result = eval(result);
     if (debug == 5 || debug == "on") {
          console.log(urld);
     }
});
```

- Callback events replace concurrency

# Asynchronous Events

```
req.on('done', result => {
    result = '' + result;
    result = eval(result);
    if (debug == 5 || debug == "on") {
        console.log(urld);
    }
});
```

- Callback events replace concurrency
- Difficult to decide when a program has terminated

# Asynchronous Events

```
req.on('done', result => {
    result = '' + result;
    result = eval(result);
    if (debug == 5 || debug == "on") {
        console.log(urld);
    }
});
```

- Callback events replace concurrency
- Difficult to decide when a program has terminated
- Don’t replay in the same process

# Dynamic Type System

```
if (sanitize) {
    req.on('data', x => '' + y);
}

req.on('done', result => {
    result = '' + result;
    result = eval(result);
    if (debug == 5 || debug == "on") {
        console.log(urld);
    }
});
```

# Dynamic Type System

```
if (sanitize) {
    req.on('data', x => '' + y);
}

req.on('done', result => {
    result = '' + result;
    result = eval(result);
    if (debug == 5 || debug == "on") {
        console.log(urld);
    }
});
```

Symbolic type coercions should be minimized as much as possible

# Dynamic Type System

```
if (sanitize) {
    req.on('data', x => '' + y);
}

req.on('done', result => {
    result = '' + result;
    result = eval(result);
    if (debug == 5 || debug == "on") {
        console.log(urld);
    }
});
```

Symbolic type coercions should be minimized as much as possible

All types need to be explored

# Strings & Regular Expressions

- JavaScript has built-in regular expressions

```
const match = /(--sanitize )?(.*)/.exec(...);
```

- Widespread Usage

| Feature | Count | % |
|---|---|---|
| Total Packages | 415,487 | 100.00% |
| Packages with regular expression | 145,100 | 34.92% |
| Packages with captures | 84,972 | 20.45% |
| Packages with no source files | 33,757 | 8.10% |
| Packages with backreferences | 15,968 | 3.84% |
| Packages with quantified backreferences | 503 | 0.12% |

# Strings & Regular Expressions (2)

```
const match = /(--sanitize )?(.*)/.exec(...);
```

- SMT Solvers support classical regular expressions
- Language extensions non-regular
- Matching precedence now matters

# Strings & Regular Expressions (3)

- Encode as classic regular expressions and string constraints

```
const match = /(--sanitize )?(.*)/.exec(...);
```

# Strings & Regular Expressions (3)

- Encode as classic regular expressions and string constraints

```
const match = /(--sanitize )?(.*)/.exec(...);
```

```
c1 in /--sanitize/ || c1 == 'undefined'
```

# Strings & Regular Expressions (3)

- Encode as classic regular expressions and string constraints

```
const match = /(--sanitize )?(.*)/.exec(...);
```

```
    c1 in /--sanitize/ || c1 == 'undefined'
    c2 in /.*/
```

# Strings & Regular Expressions (3)

- Encode as classic regular expressions and string constraints

```
const match = /(--sanitize )?(.*)/.exec(...);
```

```
s = c1 + c2 where
    c1 in /--sanitize/ || c1 == 'undefined'
    c2 in /.*/
```

# Strings & Regular Expressions (4)

- Matching precedence can cause issues

```
const match = /(--sanitize )?(.*)/.exec(...);
```

```
c1 == undefined => !c2.startsWith('--sanitize ')
```

# Strings & Regular Expressions (4)

- Matching precedence can cause issues

```
const match = /(--sanitize )?(.*)/.exec(...);
```

```
c1 == undefined => !c2.startsWith('--sanitize ')
```

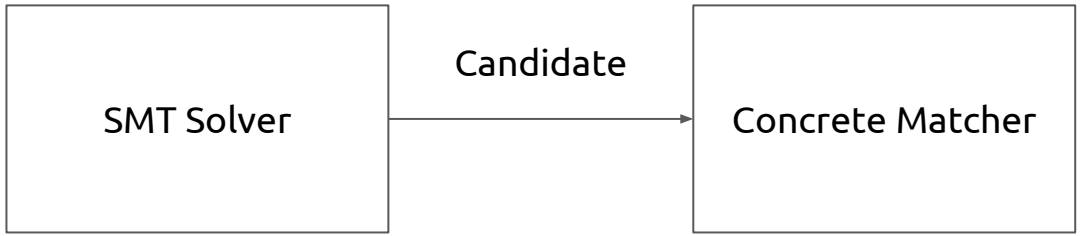
SMT Solver

# Strings & Regular Expressions (4)

- Matching precedence can cause issues

```
const match = /(--sanitize )?(.*)/.exec(...);
```

```
c1 == undefined => !c2.startsWith('--sanitize ')
```



# Strings & Regular Expressions (4)

- Matching precedence can cause issues

```
const match = /(--sanitize )?(.*)/.exec(...);
```

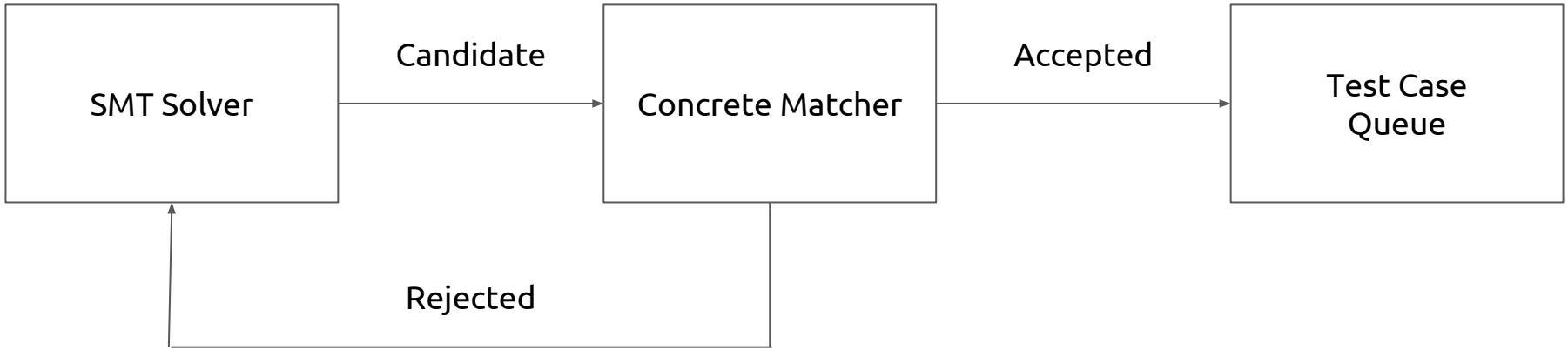
```
c1 == undefined => !c2.startsWith('--sanitize ')
```

# Strings & Regular Expressions (4)

- Matching precedence can cause issues

```
const match = /(--sanitize )?(.*)/.exec(...);
```

```
c1 == undefined => !c2.startsWith('--sanitize ')
```



# Conclusion

**JavaScript analysis is hard!**

Take care with type coercion

Keep program structure in mind

**Open Source:** ExpoSE, Z3JS, and Regex available at

https://github.com/ExpoSEJS/