Synode: Understanding and Automatically Preventing Injection Attacks on Node.js

Cristian-Alexandru Staicu\textsuperscript{1} \quad Michael Pradel\textsuperscript{1} \\
Ben Livshits\textsuperscript{2}

\textsuperscript{1}TU Darmstadt \\
\textsuperscript{2}Imperial College London, Brave Software

February 20, 2018
This Talk

Node.JS and Injections

Empirical Study

Synode

Evaluation
This Talk

Node.JS and Injections

Empirical Study

Synode

Evaluation
Node.js 101

![Diagram showing JS application and JS engine]

- **JS application**
- **JS engine**
Node.js 101

- JS application
- JS engine
- Node.JS bindings
- OS

Arrows indicate communication paths with labels 'fs, exec'.
Node.js 101

JS application

<table>
<thead>
<tr>
<th>JS engine</th>
<th>Node.JS bindings</th>
<th>OS</th>
</tr>
</thead>
</table>

fs, exec

npm

Node Package Manager
Node.js 101

JS application

JS engine

Node.JS bindings

fs, exec

OS

Node Package Manager

Node Security Project
Typical Node.JS Application

Node.JS application

- Templates engine
- Strings utility
- DB access
- Headers parser
- Vulnerable module
function backupFile(name, ext) {
var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/.localBackup/");
    
    exec(cmd.join(" "));
}
function backupFile(name, ext) {
    var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/\.localBackup/");

    exec(cmd.join(" "));
}

Malicious Payload

backupFile("-h && rm -rf * && echo ", "")
This Talk

Node.JS and Injections

Empirical Study

Synode

Evaluation
npm Codebase

236,337 packages

816,840,082 lines of JavaScript code

7,685 number of packages containing exec

2.471 average number of package dependences

>40,000 C files

9,110 number of packages containing eval

February 2016
Dependences on Injection APIs

Percentage of npm modules

- exec
- eval
- exec-level-1
- eval-level-1
- exec-level-2
- eval-level-2
- total-level-2
Data Passed to Injection APIs

Manual inspection of 150 call sites

<table>
<thead>
<tr>
<th>eval</th>
<th>exec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
</tr>
<tr>
<td>code loading</td>
<td></td>
</tr>
<tr>
<td>JSON</td>
<td></td>
</tr>
<tr>
<td>higher-order fct.</td>
<td></td>
</tr>
<tr>
<td>property read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>simple OS command</td>
</tr>
<tr>
<td></td>
<td>piped commands</td>
</tr>
<tr>
<td></td>
<td>local script</td>
</tr>
</tbody>
</table>

contain user-controlled data, out of which:
90% perform no check on this data
9% use regular expressions
Manual inspection of **150** call sites

<table>
<thead>
<tr>
<th></th>
<th>eval</th>
<th>exec</th>
</tr>
</thead>
<tbody>
<tr>
<td>code loading</td>
<td>0% 10% 20% 30%</td>
<td>0% 20% 40% 60%</td>
</tr>
<tr>
<td>JSON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>higher-order fct.</td>
<td>58%</td>
<td></td>
</tr>
<tr>
<td>property read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

58% contain user-controlled data, out of which:

- 90% perform no check on this data
- 9% use regular expressions
## Submitted Bug Reports

<table>
<thead>
<tr>
<th>Affected module</th>
<th>Confirmed</th>
<th>Time until fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mixin-pro</td>
<td>yes</td>
<td>1 day</td>
</tr>
<tr>
<td>modulify</td>
<td>no</td>
<td>–</td>
</tr>
<tr>
<td>proto</td>
<td>yes</td>
<td>155 days*</td>
</tr>
<tr>
<td>mongoosify</td>
<td>yes</td>
<td>73 days</td>
</tr>
<tr>
<td>summit</td>
<td>yes</td>
<td>–</td>
</tr>
<tr>
<td>microservicebus.node</td>
<td>yes</td>
<td>–</td>
</tr>
<tr>
<td>mobile-icon-resizer</td>
<td>yes</td>
<td>2 days</td>
</tr>
<tr>
<td>m-log</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>mongo-edit</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>mongo-parse</td>
<td>yes</td>
<td>–</td>
</tr>
<tr>
<td>mock2easy</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>mongui</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>m2m-supervisor</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>nd-validator</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>nameless-cli</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>node-mypeople</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>mongoosemask</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>kmc</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>mod</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>growl</td>
<td>yes</td>
<td>–</td>
</tr>
</tbody>
</table>

– indicates a lack response and * an incomplete fix
Lessons Learned

multiple dependences
on average each module has 2.5 direct dependences

no sanitization
only 9% use sanitization, often broken

unresponsive developers
within six months only 25% of the issues were fixed
This Talk

Node.JS and Injections

Empirical Study

Synode

Evaluation
Safe Use of Modules with Synode

- templates engine
- strings utility
- DB access
- Node.JS application
- headers parser
- vulnerable module
Overview of Synode

npm module → Static analysis
Overview of Synode

npm module → Static analysis → Statically safe programs → Safe behavior
Overview of Synode

1. npm module
2. Static analysis
3. Templates
4. Program rewriting
5. List of safe nodes
6. Statically safe programs
7. Safe behavior

Dynamic policy enforcement
Runtime inputs
Statically safe programs
Safe behavior
Overview of Synode

npm module → Static analysis → Templates → Program rewriting

List of safe nodes → Dynamic policy enforcement → Safe behavior

Statically safe programs
1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime
1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

```javascript
function backupFile(name, ext) {
    var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/.localBackup/");
    exec(cmd.join(" "));
}
```
1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

```javascript
function backupFile(name, ext) {
  var cmd = [];
  cmd.push("cp");
  cmd.push(name + "." + ext);
  cmd.push("~/.localBackup/");
  exec(cmd.join(" "));
}
```
Static Phase

1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

```javascript
function backupFile(name, ext){
    var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/localBackup/");
    exec(cmd.join(" "));
}
```

"$hole"

"~/localBackup/

"$name.$ext ~/localBackup/"
1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

```javascript
function backupFile(name, ext) {
  var cmd = [];
  cmd.push("cp");
  cmd.push(name + "." + ext);
  cmd.push("~/.localBackup/");
  exec(cmd.join(" "));
}
```
2. Synthesize runtime policy using templates:
   
   - Enforce structure via partial AST
   - For unknown parts allow only safe nodes
Static Phase

2. Synthesize runtime policy using templates:
   - Enforce structure via partial AST
   - For unknown parts allow only safe nodes

"cp $name.$ext ~/.localBackup"
Enforce policy on strings passed to injection APIs

**Policy:**

```
command
  
  literal
    value: cp

  list
    value
    value: list

  literal
    value: file.txt

  literal
    value: ~/localBackup
```
Enforce policy on strings passed to injection APIs

**Policy:**

```
command
  └── args
      ├── literal
      │    └── value
      │        └── cp
      └── list
            └── literal
                └── value
          └── '/'
                  └── localBackup
```

**Runtime string:**

```
"cp file.txt ~/localBackup"
```
Runtime Phase

Enforce policy on strings passed to injection APIs

Policy:

```
command
  |--- args
  |    |
  |    |
literal
  |--- list
  |    |
  |    |
literal
  |--- literal
  |    |
  |    |
value
  |--- cp
  |    
  |    |
value
  |--- ??
  |    
  |    
value
  |--- ~/.localBackup
```

Runtime string:

```
"cp file.txt ~/localBackup"
```
Runtime string:
"cp x || rm * -rf ~/.localBackup"
Runtime Phase

Runtime string:
"cp x || rm * -rf ~/.localBackup"
This Talk

Node.JS and Injections

Empirical Study

Synode

Evaluation
Evaluation: Static Phase

Setup
- 51K call sites of injection APIs

Precision
- 36.7% of the call sites statically safe
- 63.3% to be checked at runtime

Context
- most call sites have at least:
  - 10 constant characters per template
  - 1 unknown per template

Performance
- 4.4 seconds per module
Evaluation: Runtime Phase

Setup
- 24 modules
- 56 benign and 65 malicious inputs

Results
- zero malicious inputs that we do not stop
- five benign inputs that we incorrectly stop
- overhead: 0.74 milliseconds per call
Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react
Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react
Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react

Prevention of injections

- Automatic and easy to deploy
  - https://github.com/sola-da/Synode
- Small overhead and high accuracy
Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react

Prevention of injections

- Automatic and easy to deploy
  [https://github.com/sola-da/Synode](https://github.com/sola-da/Synode)
- Small overhead and high accuracy

Open challenges

- More precise static analysis
- Automatic generation of attacks
Example Limitation: Array.map()

```javascript
var keys = Object.keys(dmenuOpts);
var dArgs = keys.map(function(flag) {
    return '−' + flag + ' "' + dmenuOpts[flag] + '"';
}).join(' ');

var cmd = 'echo | dmenu −p "Password:" ' + dArgs;
exec(cmd);
```

Inferred template

```
'echo | dmenu −p "Password:" $dArgs'
```
Implementation

- Intraprocedural static analysis
- Based on **Google Closure Compiler**
- Policy for unknown parts:
  - `exec`: literal
  - `eval`: literal, identifier, property, array expression, object expression, member expression, expression statement
Beyond eval and exec

- `vm.runInThisContext()`

```javascript
var vm = require('vm');
vm.runInThisContext(
  "console.log('" + input + ");");
```

- `execa` module (1,000 dependents)

```javascript
module.exports.shell = function(cmd) {
  args = ['-c', cmd]
  childProcess.spawnSync("/bin/sh", args);
}
```
Why is the Application Domain Unique?

20 out of 66 advisories are injections (Node Security Project)

- Bad habits
- Unnecessary code reuse (see left-pad)
- No sandbox