TAKING SECURITY SERIOUSLY

DR. PHILIPPE DE RYCK

https://Pragmatic Web Security.com

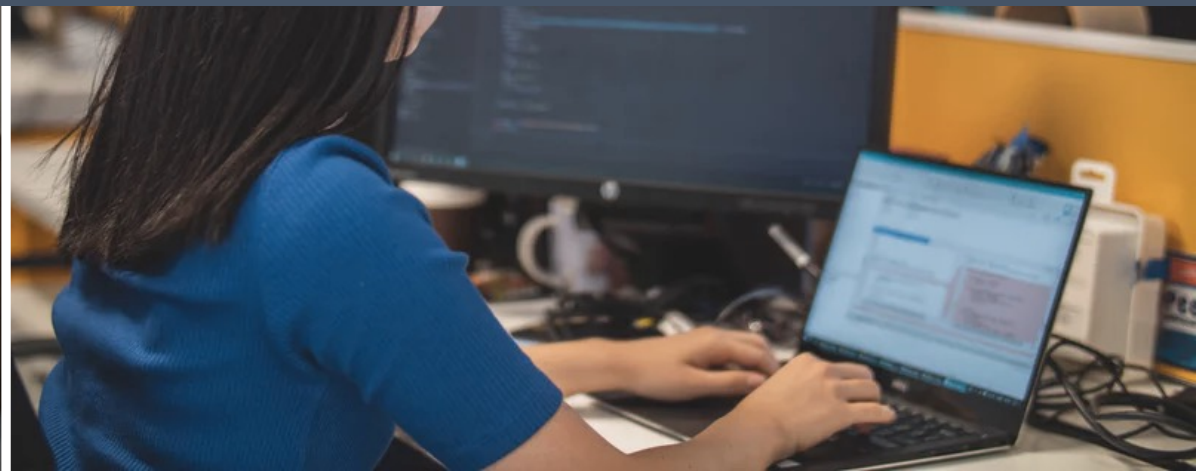
KrebsonSecurity
In-depth security news and investigation

We Take Your Privacy and Security. Seriously.

September 29, 2014 135 Comments

"Please note that [COMPANY NAME] takes the security of your personal data very seriously." If you've been on the Internet for any length of time, chances are very good that you've received at least one breach notification email or letter that includes some version of this obligatory line. But as far as lines go, this one is about as convincing as the classic break-up line, "It's not you, it's me."

WHAT DOES IT MEAN TO TAKE SECURITY SERIOUSLY?



I am *Dr. Philippe De Ryck*

Pragmatic Web Security
Security for developers

Founder of Pragmatic Web Security

Google Developers Experts

Google Developer Expert

AMBASSADOR PROGRAM

Auth0 Ambassador

Secure Application Development

SecAppDev organizer

I help developers with security

- **Hands-on in-depth security training**
- **Advanced online security courses**
- **Security advisory services**

https://pragmaticwebsecurity.com

Giving the browser a snippet of code mixed with data

```
1  let data = "<p>" + review + "</p>"
2  document.getElementById("msg").innerHTML = data
```

innerHTML relies on the browser's code parser to handle the data

A review submitted by a malicious user

```
1  This restaurant is highly recommended. The food
2  is exquisite and the service is impeccable.<img
3  src="none.png" onerror="alert('Go there, now!')">
```

Giving the browser a snippet of code mixed with data

```
1  let data = "<p>" + review + "</p>"
2  document.getElementById("msg").innerHTML = data
```

innerHTML relies on the browser's code parser to handle the data

Giving the browser code and data with context information

```
1  let p = document.createElement("p")
2  p.textContent = review
3  document.getElementById("msg").appendChild(p)
```

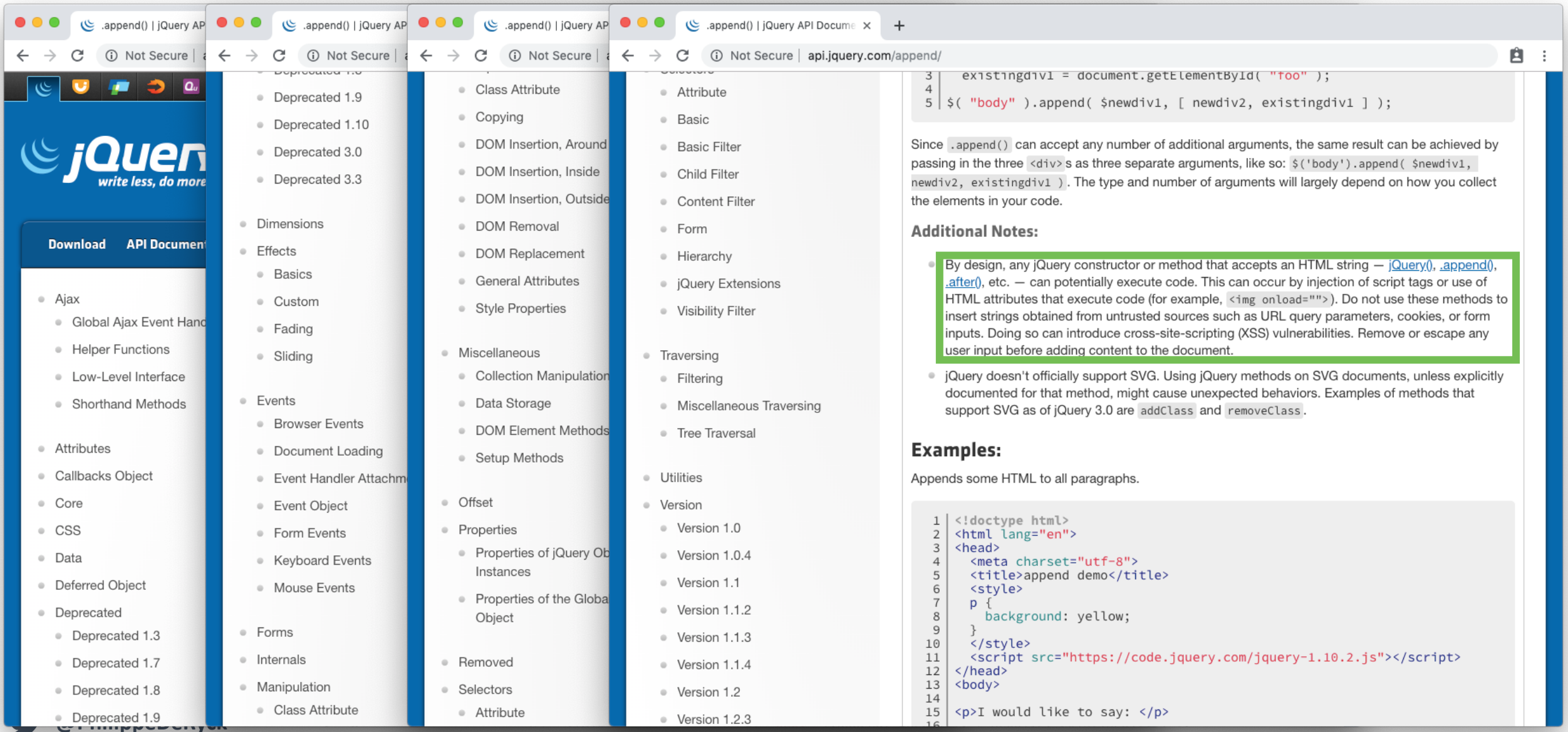
Using the proper DOM APIs provides the browser with context, avoiding the confusion that leads to XSS

A review submitted by a malicious user

```
1  This restaurant is highly recommended. The food
2  is exquisite and the service is impeccable.<img
3  src="none.png" onerror="alert('Go there, now!')">
```

Appending a div *with jQuery's* append *function*

```
1  $("#reviews").append(`<p class="review">${review}</div>`);
```



UNDERSTAND YOUR LIBRARIES

Libraries and frameworks are not always secure out of the box. Understanding how your library of choice handles security is crucial.

One framework.

Mobile & desktop.

GET STARTED

{{myDirtyData}}

An Angular template template to combine data with HTML

```
1  <div>
2    <h3>{{ review.title }}</h3>
3    <p>{{ review.content }}</p>
4  </div>
```

By default, Angular escapes values embedded in a template before rendering them

A review submitted by a malicious user

```
1  This restaurant is <b>highly recommended</b>. The
2  food is exquisite and the service is impeccable. <a
3  href="https://pics.example.com">Check out my story
4  here!</a><img src="none.png" onerror="alert('Go
5  there, now!')">
```

https://restograde.com

Absolutely awesome

This restaurant is highly recommended. The food is exquisite and the service is impeccable. Check out my story here!<img src="none.png" onerror="alert('Go there, now!')">

An Angular template to render user-provided HTML

```
1  <div>
2    <h3>{{ review.title }}</h3>
3    <p [innerHTML]="review.content"></p>
4  </div>
```

A review submitted by a malicious user

```
1  This restaurant is <b>highly recommended</b>. The
2  food is exquisite and the service is impeccable. <a
3  href="https://pics.example.com">Check out my story
4  here!</a><img src="none.png" onerror="alert('Go
5  there, now!')">
```

[innerHTML] does not directly expose *innerHTML* property, but sanitizes the data first

Binding HTML into the page with Angular is secure by default

```
trustAsHtml(myDirtyData)
```

~~`trustAsHtml(myDirtyData)`~~

`bypassSecurityTrustHtml(myDirtyData)`

Make insecurity explicit

Explicitly marking features as insecure
helps prevent accidental misuse
and simplifies code scanning efforts

Data containing HTML with style attributes

```
1 <p style="color: red">…</p>
```

The Angular sanitizer is secure-by-default and does not allow the use of *style* attributes

The only way to allow style attributes is to bypass sanitization

```
1  <div [innerHTML]="sanitizer.bypassSecurityTrustHtml(data)"></div>
```

This code completely disables the sanitizer, creating a massive vulnerability in the application

An Angular pipe that sanitizes HTML but allows style information in attributes

```
1  @Pipe({
2    name: 'sanitizeWithStyle'
3  })
4  export class SanitizeWithStylePipe implements PipeTransform {
5
6    constructor(private sanitizer : DomSanitizer) {}
7
8    transform(html: string) : SafeHtml {
9    // Allowing CSS is still not recommended
10   return this.sanitizer.bypassSecurityTrustHtml(
11           DOMPurify.sanitize(html, {ADD_ATTR: ['style']}));
12   }
13 }
```

This use of *bypassSecurityTrustHtml* can be marked as checked, so code scanning tools ignore it

DOMPurify sanitizes the data, but is configured to allow *style* attributes

The application code no longer calls bypassSecurityTrustHtml *directly*

```
1  <div [innerHTML]="data | sanitizeWithStyle"></div>
```

Allowing style information can still result in attacks, so use a pattern like this with care

Setup developers for security success

Secure-by-default frameworks reduce the need for knowledge

Subtle security nudges reduce the risk of mistakes

Encapsulate dangerous functions and use linting to prevent direct usage

A typical DOM-based XSS vulnerability, which also occurs in many script gadgets

```
1  document.querySelector("#data").innerHTML = "<b>Hello world!</b>" + data;
```

Providing data and code to the browser's HTML parser triggers DOM-based XSS

A page enabling a **Trusted Types** *policy in a response header*

```
1  Content-Security-Policy: require-trusted-types-for 'script'
2
3  document.querySelector("#data").innerHTML = "<b>Hello world!</b>" + data;
```

Enabling Trusted Types eliminates an entire class of XSS vulnerabilities

When *Trusted Types* is enabled, text-to-code sinks like *innerHTML* throw a TypeError

Trusted Types does not affect the use of proper DOM APIs

```
1  Content-Security-Policy: require-trusted-types-for 'script'
2
3  let msg = document.createElement("span");
4  msg.setAttribute("class", "italic");
5  msg.innerText = e.data;
6  document.getElementById("msg").appendChild(msg);
```

InnerHTML can still be used, as long as it is assigned a Trusted Type value

```
1  Content-Security-Policy: require-trusted-types-for 'script'
2
3  let safeData = DOMPurify.sanitize("<b>Hello world!</b>" + data, {RETURN_TRUSTED_TYPE: true});
4  document.querySelector("#data").innerHTML = safeData;
```

The safe value can now be assigned to *innerHTML*, because it is properly sanitized by DOMPurify

DOMPurify has built-in support for *Trusted Types*, and can be instructed to return the data as a *TrustedHTML* value instead of a string

Trusted Types for DOM manipulation

- UNOFF

Usage % of all users ?

Global 70.93%

An API that forces developers to be very explicit about their use of powerful DOM-injection APIs. Can greatly improve security against XSS attacks.

Current aligned | Usage relative | Date relative | Filtered | All

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *
	12-81		4-81		10-68			
6-10	83-92	2-91	83-92	3.1-14	69-77	3.2-14.4		2.1-4.4.4
11	93	92	93	14.1	78	14.7	all	93
		93-94	94-96	15-TP				

Enable trusted types by setting a CSP policy

```
1  Content-Security-Policy: trusted-types angular; require-trusted-types-for 'script'
```

Enable Angular's built-in Trusted Types policy

Tells the browser to only allow trusted types in the DOM

Angular works as before, but the browser no longer allows insecure behavior (e.g., bypassSecurityTrustHtml, native innerHTML)

The browser now refuses to assign unsafe content to innerHTML

```
1  this.div.nativeElement.innerHTML = this.inputValue;
```

```
▶Uncaught TypeError: Failed to set the 'innerHTML' property on 'Element': This document requires 'TrustedHTML' assignment.    index.js:1
    at HTMLIFrameElement.e.onload (index.js:1)
    at fe (index.js:1)
    at index.js:1
    at index.js:1
```

Enable trusted types by setting a CSP policy

```
1  Content-Security-Policy:
2      trusted-types angular angular#unsafe-bypass; require-trusted-types-for 'script'
```

Enable Angular's built-in Trusted Types policy

Allow the use of bypassSecurityTrustHtml, secure or not

The browser still refuses to assign unsafe content to innerHTML

```
1  this.div.nativeElement.innerHTML = this.inputValue;
```

```
▸Uncaught TypeError: Failed to set the 'innerHTML' property on 'Element': This document requires 'TrustedHTML' assignment.   index.js:1
    at HTMLIFrameElement.e.onload (index.js:1)
    at fe (index.js:1)
    at index.js:1
    at index.js:1
```

Use platform-level security features

Building applications on a secure platform offers a significant advantage

Trusted Types prevents dangerous DOM manipulations that lead to XSS

A secure platform not only covers the application, but also its dependencies

> 97%

of code in a modern web app are dependencies

```
$ ng new clean-app

? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? Sass


added 1169 packages from 1030 contributors and audited
42445 packages in 28.75s
```

```
$ cloc node_modules/
```

Language	files	blank	comment	code
JavaScript	12683	145344	525680	1773037
JSON	1555	104	0	161571
Markdown	1385	65564	4	157446
TypeScript	2892	9625	90588	104376
HTML	274	1656	218	33724
CSS	148	299	2301	22382
C++	75	3784	3501	22332
Python	51	4205	7606	18695
C/C++ Header	101	2758	1858	15114
LESS	482	1611	410	11321
XML	20	3237	1300	7617
YAML	163	140	112	2416
Bourne Shell	18	292	333	1500
SVG	8	2	2	776
make	30	236	39	715
Windows Module Definition	7	115	0	641
DTD	1	179	177	514
...				
SUM:	19983	239598	634354	2336228

```
$ ng new clean-app

? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? Sass


added 1169 packages from 1030 contributors and audited
42445 packages in 28.75s
```

Kevin Beaumont 🥴 

@GossiTheDog

Follow

NPM library with 2m installs has a backdoor, looks to be some kind of Trojan (stealer?)

dominictarr commented 5 days ago

Owner

he emailed me and said he wanted to maintain the module, so I gave it to him. I don't get any thing from maintaining this module, and I don't even use it anymore, and havn't for years.

👍 61 | 👎 150 | 😄 30 | 🎉 9 | 😕 24 | ❤️ 35

7:39 PM - 26 Nov 2018

40%

of packages rely on known vulnerable code*

*estimated by the authors of

Small world with high risks: a study of security threats in the npm ecosystem

bash ⌥⌘1

```
                    === npm audit security report ===

# Run  npm update handlebars --depth 7  to resolve 1 vulnerability
```

High	Prototype Pollution
Package	handlebars
Dependency of	react-scripts [dev]
Path	react-scripts > jest > jest-cli > @jest/core > @jest/reporters > istanbul-reports > handlebars
More info	https://npmjs.com/advisories/1164

```
found 1 high severity vulnerability in 905626 scanned packages
  run `npm audit fix` to fix 1 of them.
```

DEPENDENCY-CHECK

How to read the report | **Suppressing false positives** | **Getting Help: google group** | **github issues**

Project: Restograde

Scan Information (show all):

- *dependency-check version*: 4.0.0
- *Report Generated On*: Dec 5, 2018 at 09:43:02 +01:00
- *Dependencies Scanned*: 1217 (1048 unique)
- *Vulnerable Dependencies*: 10
- *Vulnerabilities Found*: 10
- *Vulnerabilities Suppressed*: 0
- ...

Equifax uses *Apache Struts 2* to build applications

a patched version of *Struts2* fixes a remote code execution vulnerability

March 7th, 2017

Equifax discovers the breach of their systems

July 29th, 2017

Equifax announces the breach

September 7th, 2017

May 2017

attackers escalate the attack to full-scale data exfiltration

March 10th, 2017

attackers start probing *Equifax* systems using the *Struts* vulnerability

78%

of vulnerabilities occur in indirect dependencies

- Pulse
- Contributors
- Traffic
- Commits
- Code frequency
- **Dependency graph**
- Alerts
- Network
- Forks

Dependency graph

Dependencies | Dependents

⚠ We found potential security vulnerabilities in your dependencies.

Dependencies defined in these manifest files have known security vulnerabilities and should be updated:

restograde-angular/package-lock.json *5 vulnerabilities found*
reviewer-angular/package-lock.json *5 vulnerabilities found*

See security alerts

Only the owner of this repository can see this message.
Learn more about vulnerability alerts

These dependencies are defined in **pws-restograde**'s manifest files, such as **reviewer-angular/package-lock.json**, **reviewer-angular/package.json**, and **restograde-angular/package-lock.json**.

Snyk for Developers & DevOps

Snyk continuously monitors your application's dependencies and lets you quickly respond when new vulnerabilities are disclosed.

Fix your vulnerabilities

- **Single click fix - generate a fix PR from UI, CLI wizard**
- Upgrade - Automatically calculates the minimal direct dependency version upgrade needed
- Precision patch - Use patches backported by Snyk security team to fix when direct upgrade is not available or it'll take time to have upgrade implemented
- Automatic fix for new vulnerabilities - Automatically generate fix pull requests for newly discovered vulnerabilities

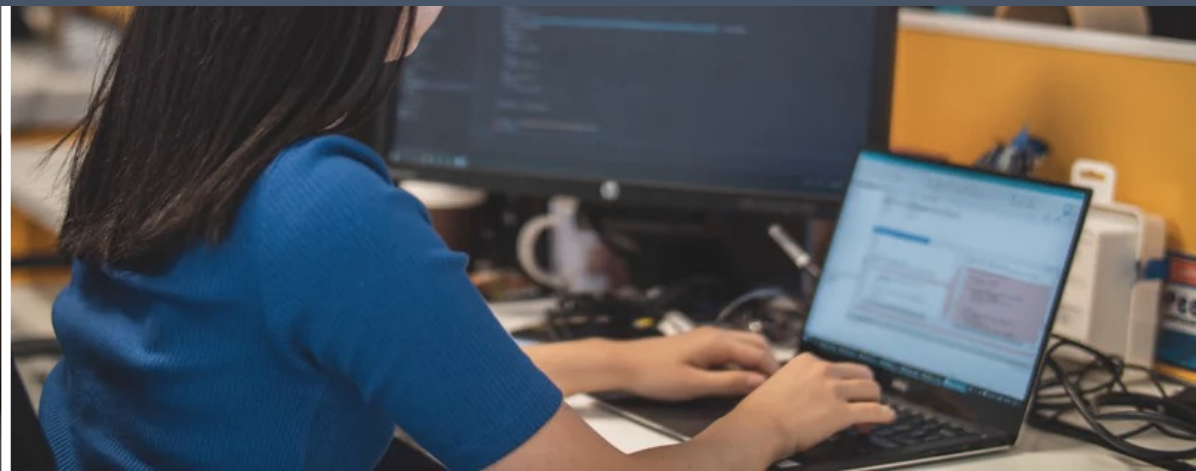
Secure your dependency graph

Setup dependency monitoring for all your projects

Patch your software, both continuously and urgently

Focus on the dependencies that matter

EMPOWER PEOPLE TO TAKE SECURITY SERIOUSLY



In-depth online courses to help you take security seriously

https://courses.pragmaticwebsecurity.com

THANK YOU!

Follow me on Twitter to stay up to date on security resources and courses

@PhilippeDeRyck

Images from Unsplash.com