Security in the UI

David Mann

@MannD | www.HeirloomSoftware.com

Topics

Understanding the Threat

- Cross-Site Scripting (XSS)

Angular Default Protection

Overriding the Defaults

XSS: Cross-Site Scripting

A vulnerability which allows attackers to inject executable content (primarily JavaScript) into web pages

~~Inject & execute random script~~

Vulnerabilities

Forms

http://...

URL

Search box

Comments

No results for your search <script>alert('pwned!')</script>

More Than Just Script Tags

<body onload="...">

<a href="javascript:...">

Dig Deeper

https://www.pluralsight.com/courses/ethical-hacking-web-servers

https://www.pluralsight.com/blog/tutorials/learning-path-ethical-hacking

Angular treats all values as untrusted by default

https://angular.io/guide/security

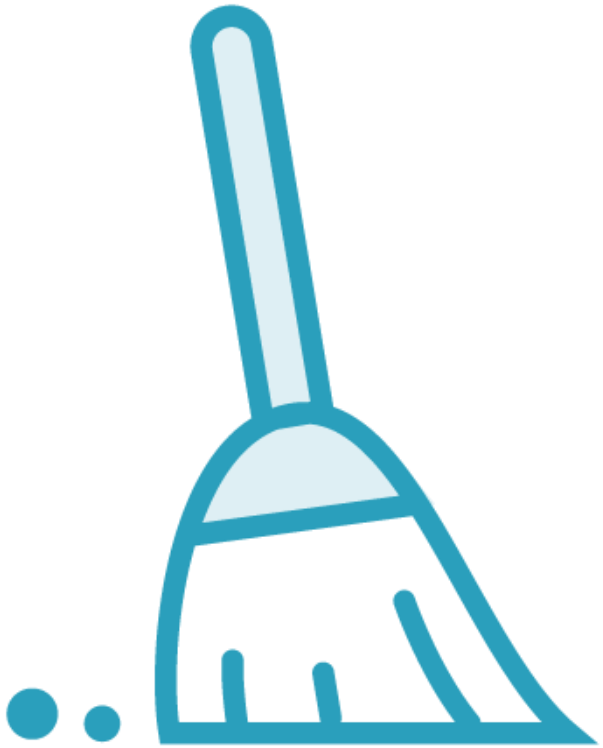
Distrust by Default

Template HTML

Binding Expressions

Attributes

Sanitization



```
<script>alert('pwned!')</script>

'<script>alert("pwned");</script>'
```

Content cleansing

Context-sensitive

Sanitization Context

HTML

URL

Style

Resource URL

HTML

```
<div [innerHtml]="htmlValue" ></div>
```

innerHtml

```
WARNING: sanitizing HTML stripped some content


"
<script>alert("pwned");</script>
"
```

URL

```
<a [href]="hrefValue">Test Link</a>
```

href

```
WARNING: sanitizing unsafe URL value javascript:alert("pwned");

▶<a _ngcontent-c6 href="unsafe:javascript:alert("pwned");">…</a>
```

Style

```
<div [style]="styleValue">
    Sample Div
</div>
```

```
styleValue = 'background-color:red';
```

style

```
WARNING: sanitizing unsafe style value background-color:red
```

```
<div _ngcontent-c6 style>
    Sample Div
</div>
```

Resource URL

```
<script [src]="srcValue"></script>
<script src='/myInjectedScript.js'>
</script>
```

<script src...

<iframe src...

Interpolation

```
{{htmlValue}}
```

```
htmlValue = `<script>
                alert("pwned");
            </script>`;
```

Always sanitized

```
"
<script>alert("pwned");</script>
"
```

```
<script>alert('pwned!')</script>
```
← interpolated

```
alert('pwned!')
```
← bound

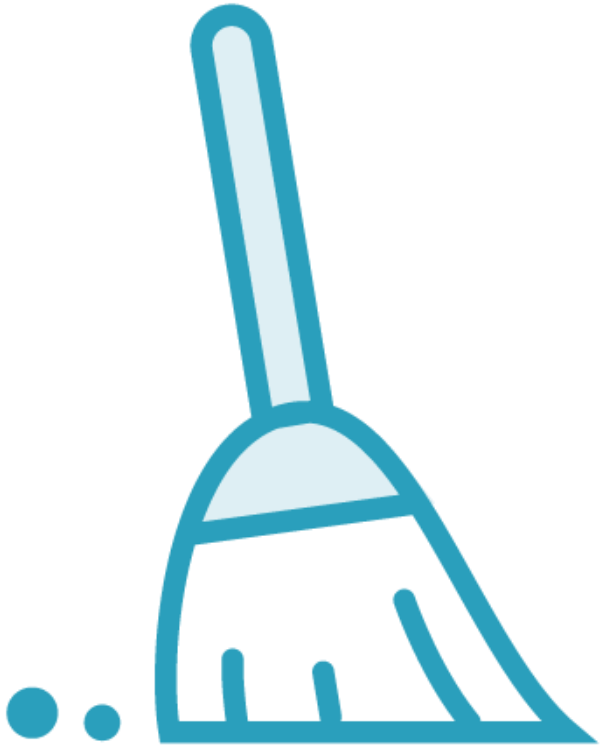
Isn’t All of That Enough?

Yes

Bypassing Angular's Protection

Change requirements?

No other solution?



DomSanitizer

```
constructor(private sanitizer: DomSanitizer)
```

bypassSecurityTrustHtml

bypassSecurityTrustScript

bypassSecurityTrustStyle

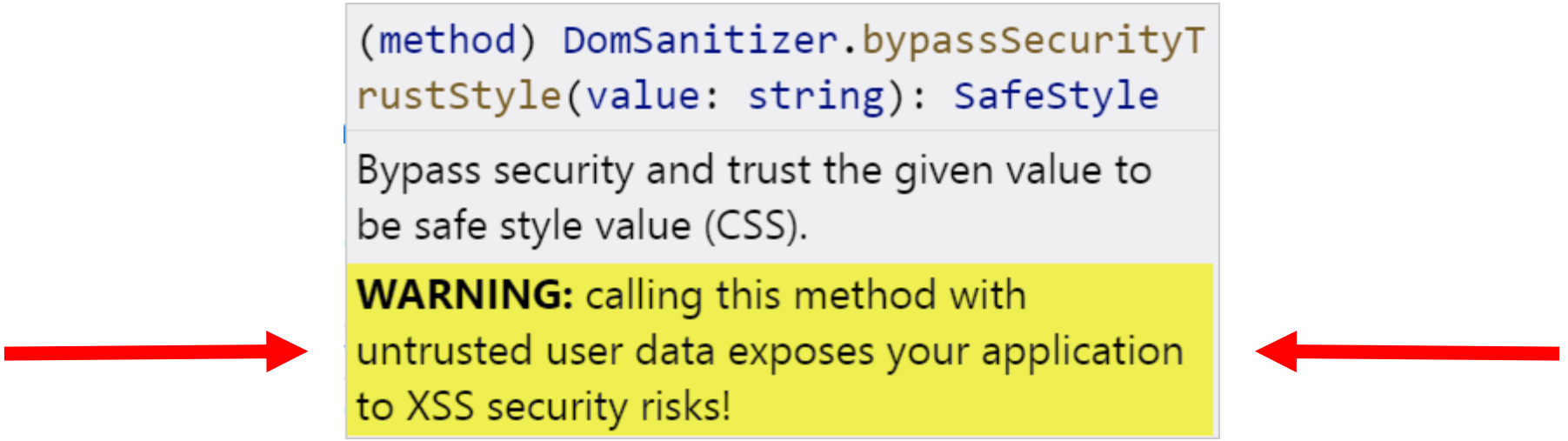
bypassSecurityTrustUrl

bypassSecurityTrustResourceUrl

```
let safeHtml: SafeHtml = sanitizer.bypassSecurityTrustHtml('unsafe content');
let safeScript: SafeScript = sanitizer.bypassSecurityTrustScript('unsafe content');
let safeStyle: SafeStyle = sanitizer.bypassSecurityTrustStyle('unsafe content');
let safeUrl: SafeUrl = sanitizer.bypassSecurityTrustUrl('unsafe content');
let safeResourceUrl: SafeResourceUrl = sanitizer.bypassSecurityTrustResourceUrl('unsafe content');
```

Safe Objects

Not Kidding Around



Key Takeaways

Cross-Site scripting
Sanitization
Be careful!

Next Up

Wrap Up