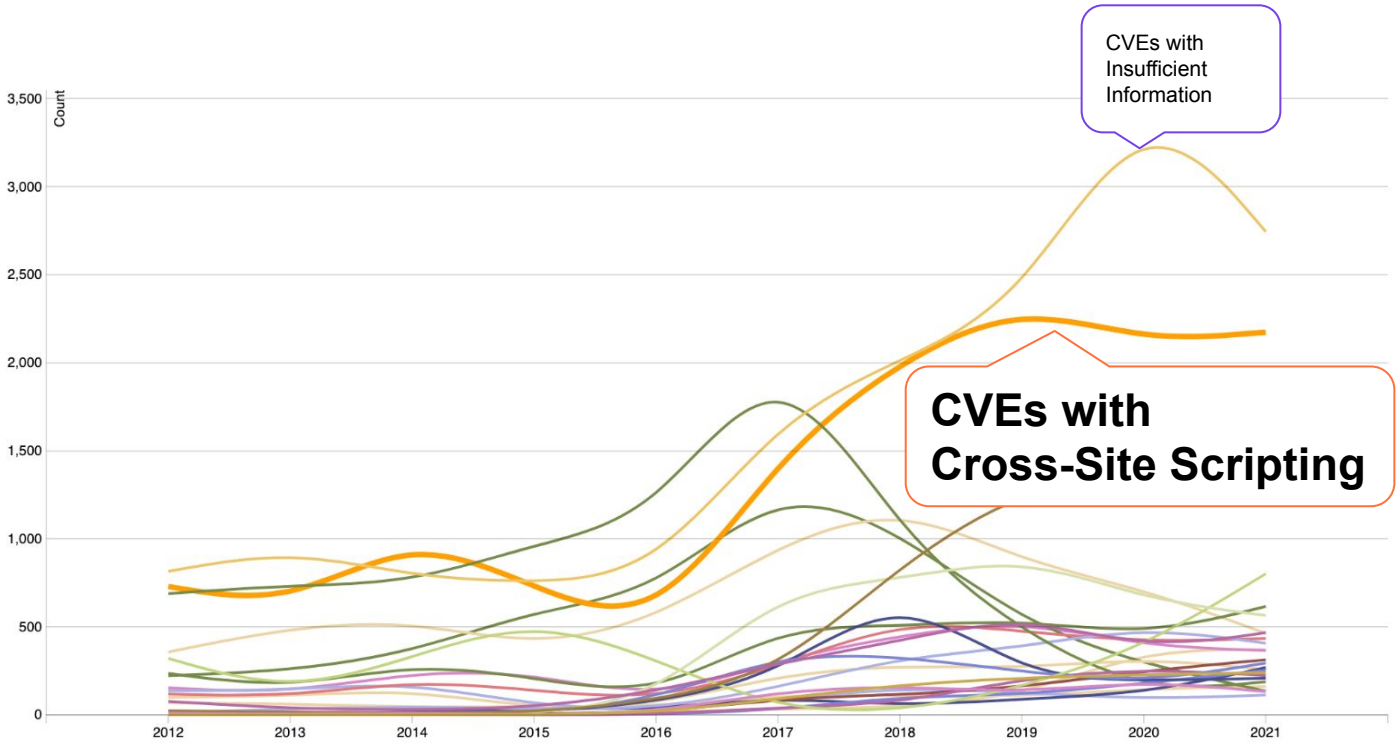
What if XSS was a browser bug?

Frederik Braun
Staff Security Engineer
freddyb@mozilla.com



[Source: https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cwe-over-time]

[Source: https://twitter.com/joernchen/status/1086237923652046849]

div.setHTML()

Agenda

1. Intro into (DOM-based) XSS
2. Browser-based XSS Defenses in the past
3. Making an HTML Sanitizer
4. The Security Considerations of a Sanitizer
5. Getting Involved

Frederik Braun
@freddy@security.plumbing
freddy@mozilla.com

Staff Security Engineer
moz://a

Subresource Integrity

X-Frame-Options: All about Clickjacking?

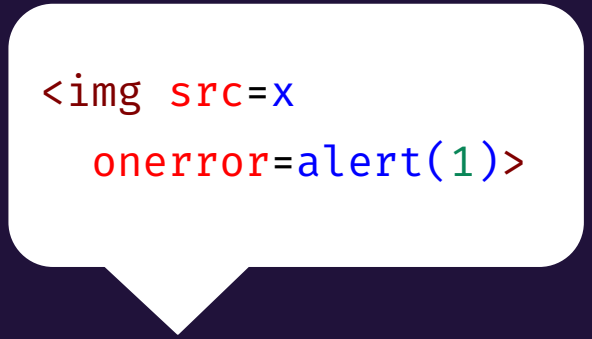
eslint-plugin-no-unsanitized

`div.innerHTML = something`

Vanilla JavaScript Web Development

```
div.innerHTML = evil
```

DOM-based XSS



div.innerHTML = evil

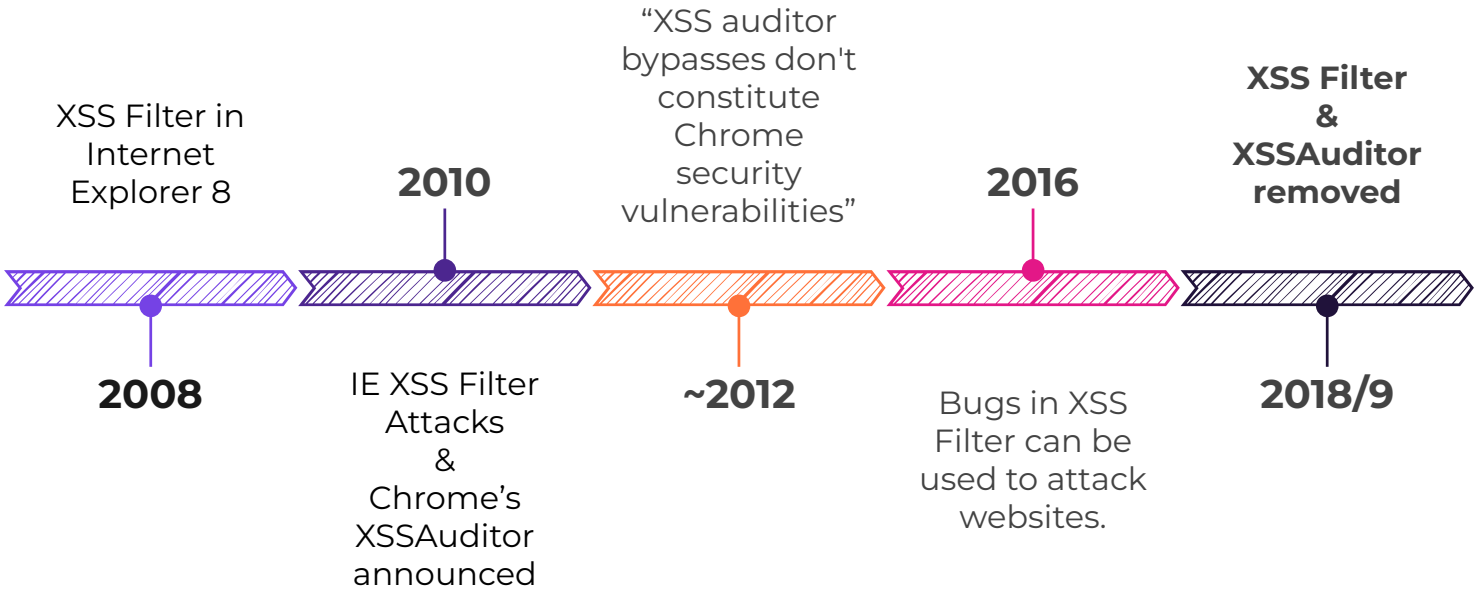
DOM-based XSS

Browser-based XSS defenses

Two case studies

Browser-based XSS ~~defenses~~ mitigations

XSS Filters



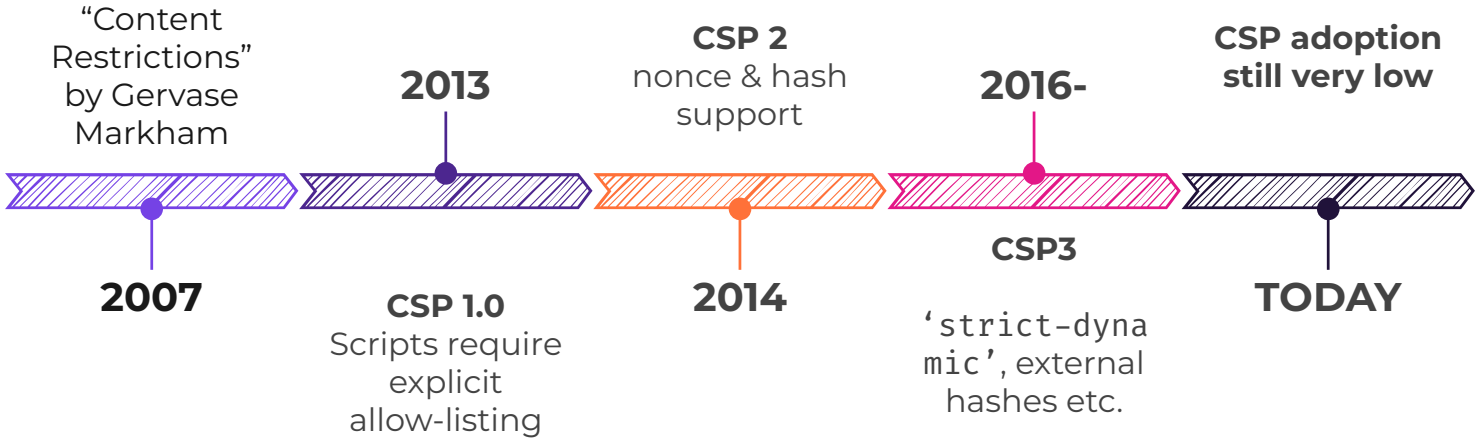
(*2008 - † 2019)

XSS Filters

“Better than RegEx” is not good enough. Cf. http://langsec.org/occupy/

You might cause more harm than good

Content Security Policy (CSP)



(*2007 - † ?)

Content Security Policy

<20% have a CSP which controls script

of those, 94% still allow inline scripts

https://almanac.httparchive.org/en/2022/security#content-security-policy

Lessons learned

Adoption must be very easy:
Low Complexity

We need *High Compatibility* with existing content

Focus on Prevention rather than Mitigation

https://almanac.httparchive.org/en/2022/security#content-security-policy

What if **DOM-based XSS** was a browser bug?

Our Goal

`"div.innerHTML = evil"`, but without XSS

Sanitizers Today

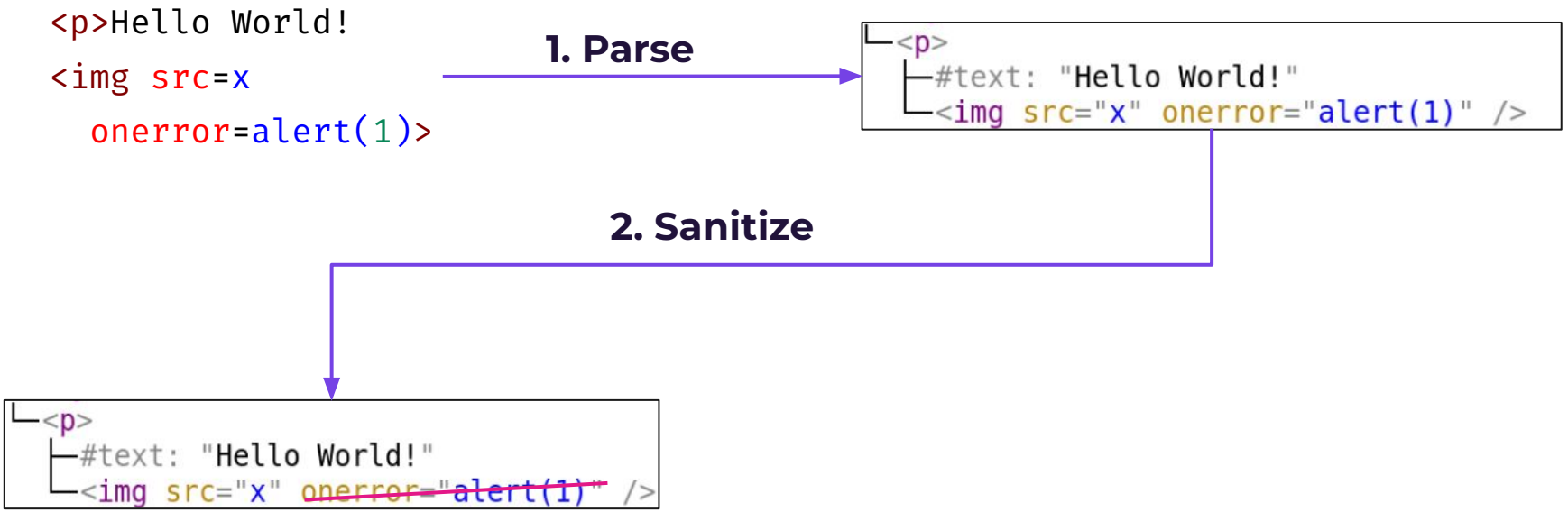
```
let clean = DOMPurify.sanitize(evil, options);

div.innerHTML = clean;
```

What's in a Sanitizer?

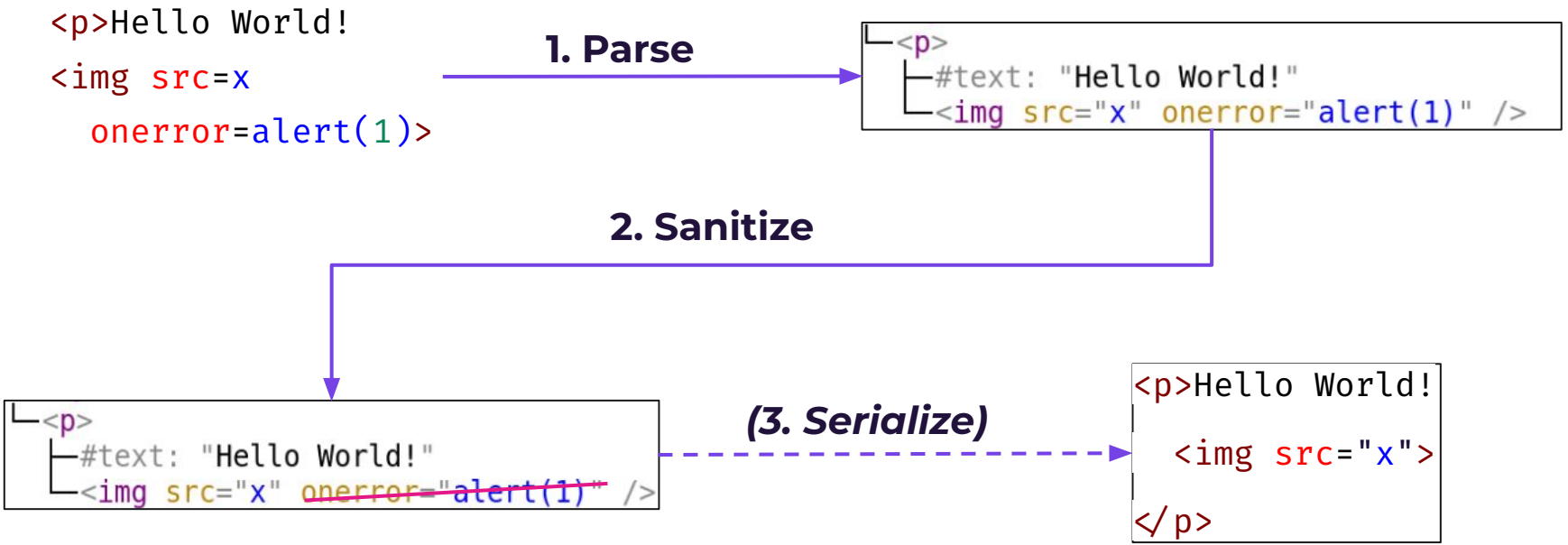
```
<p>Hello World!
<img src=x
  onerror=alert(1)>
```

1. Parse

What's in a Sanitizer?



What's in a Sanitizer?

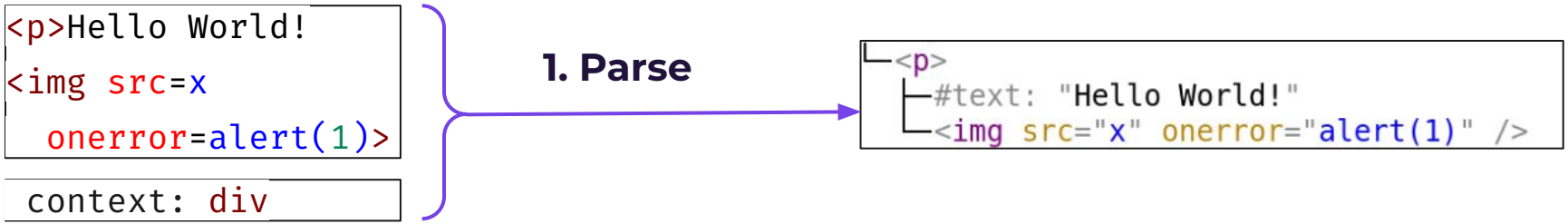


Sanitizers Today

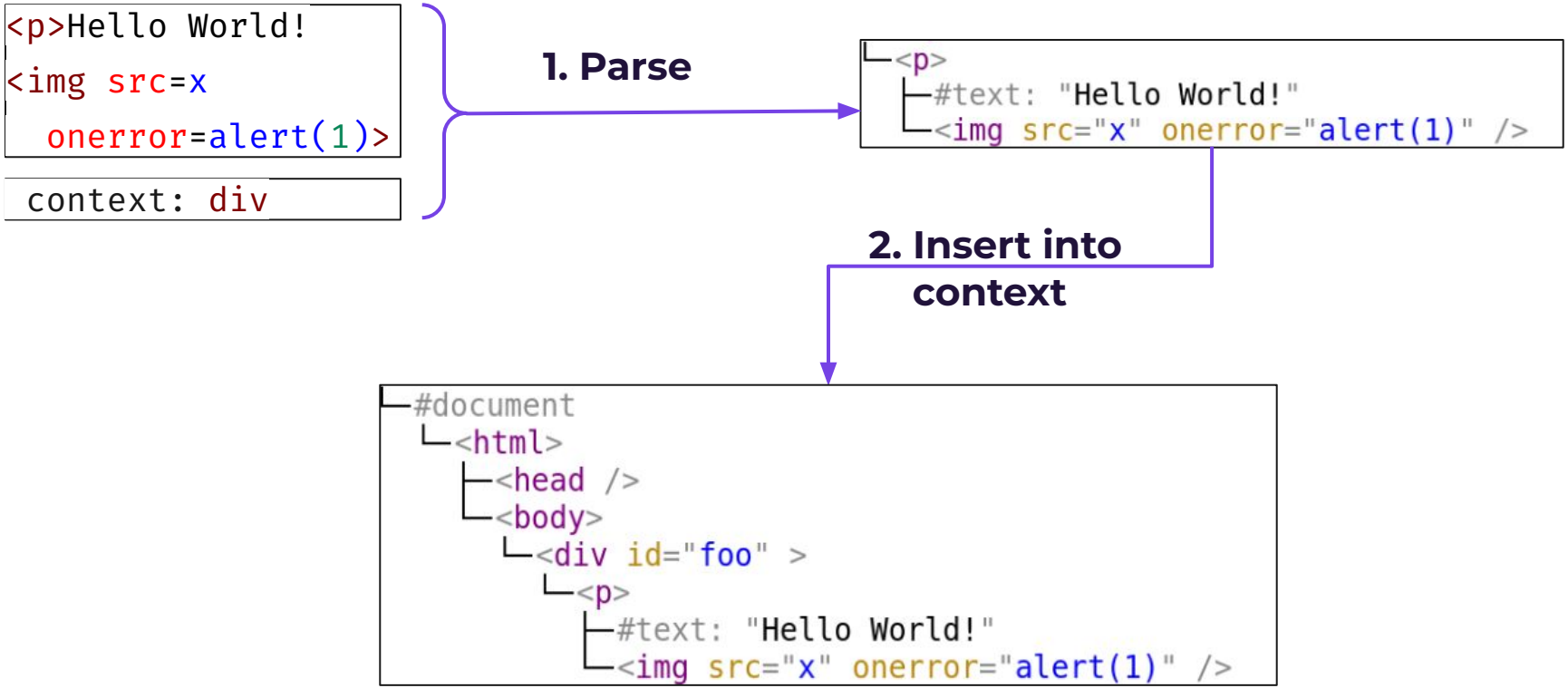
new Sanitizer(config).sanitize(

```
let clean = DOMPurify.sanitize(evil, options);

div.innerHTML = clean;
```

What's in a `div.innerHTML=` Assignment?



What's in a `div.innerHTML=` Assignment?



What’s in a Sanitizer?

1. Parse

2. Sanitize

3. Serialize

We’re parsing TWICE now?

API: Revision 1

```
mySanitizer = new Sanitizer(options)

mySanitizer.sanitize() // DocFragment
```

```
div.append(
  mySanitizer.sanitize(evil)
)
```

Improving with Feedback

Looking for Bugs here. Anyone got some bugs?

Sanitizer is less expressive than `innerHTML`

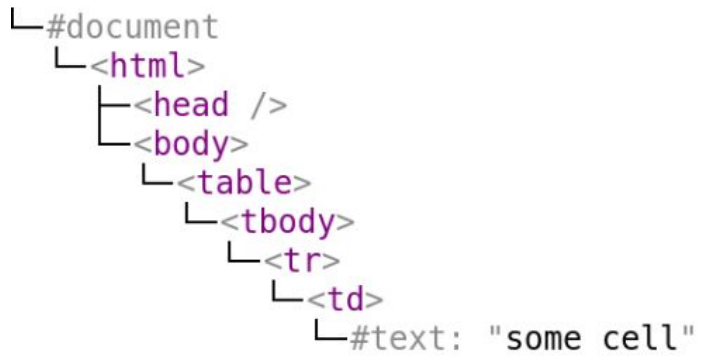
01

https://github.com/WICG/sanitizer-api/issues/42
Reported by Anne van Kesteren (@annevk)

innerHTML

Without the Sanitizer

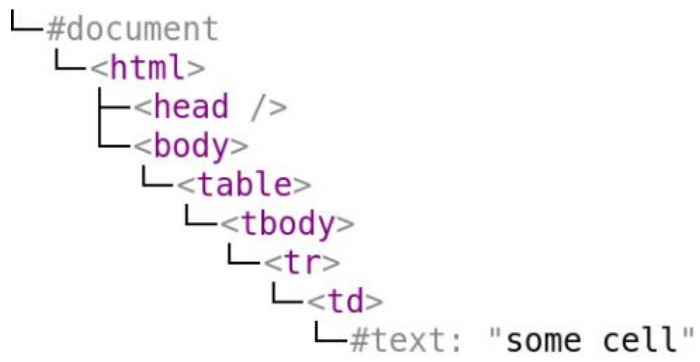
```
tableElement.innerHTML =
  "<tr><td>some cell</td></tr>"
```



innerHTML

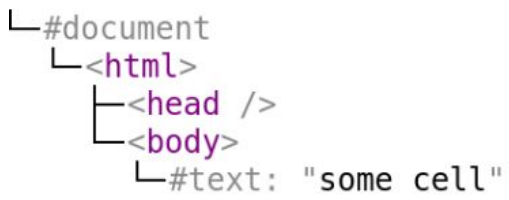
Without the Sanitizer

```
tableElement.innerHTML =
  "<tr><td>some cell</td></tr>"
```



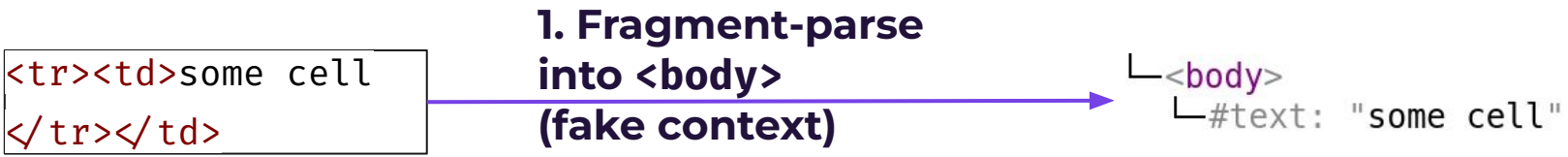
With the Sanitizer

```
tableElement.append(
  mySanitizer.sanitize(sameInput))
```



HTML Parsing is *contextual*

Fragment parsing without context



Sanitizer Bypass with `iframe srcdoc`

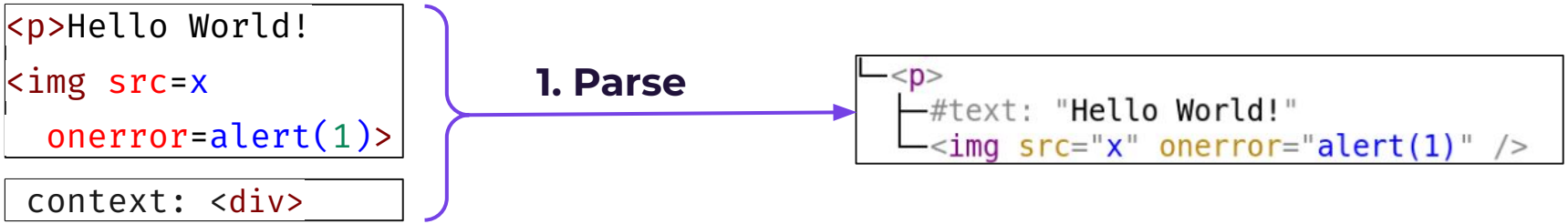
02

https://bugzilla.mozilla.org/show_bug.cgi?id=1669945
Reported by Michał Bentkowski (@SecurityMB)

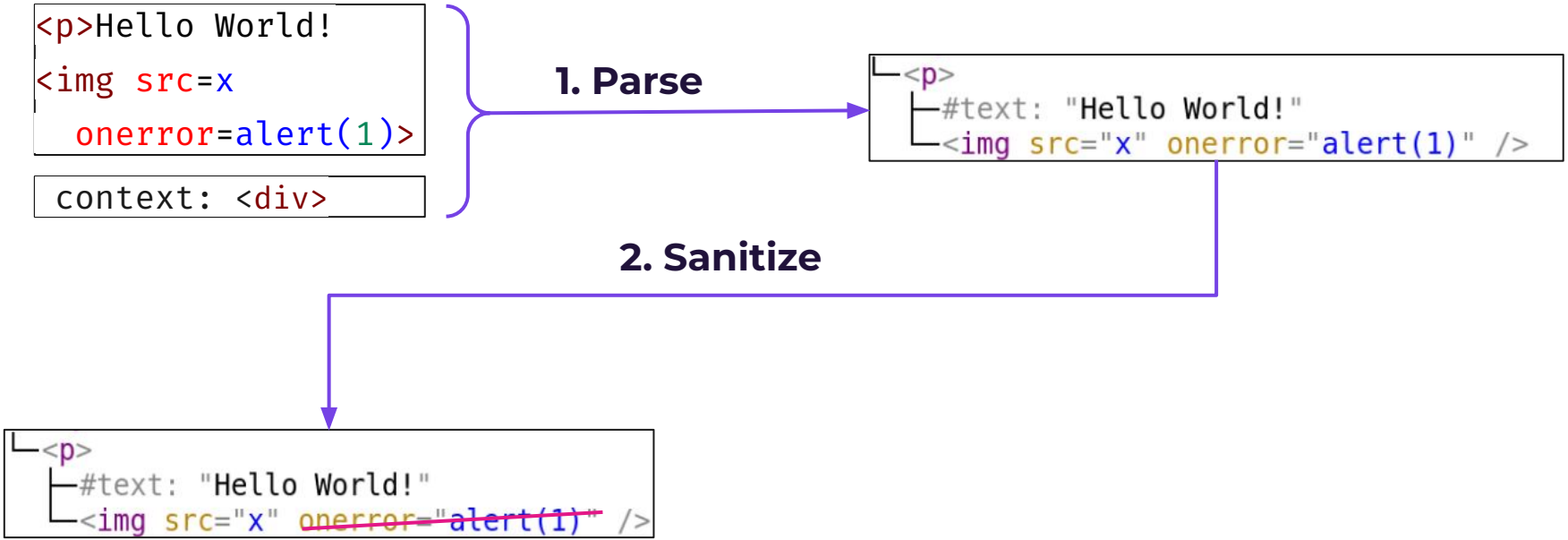
Burn all Parsers!

@    # !

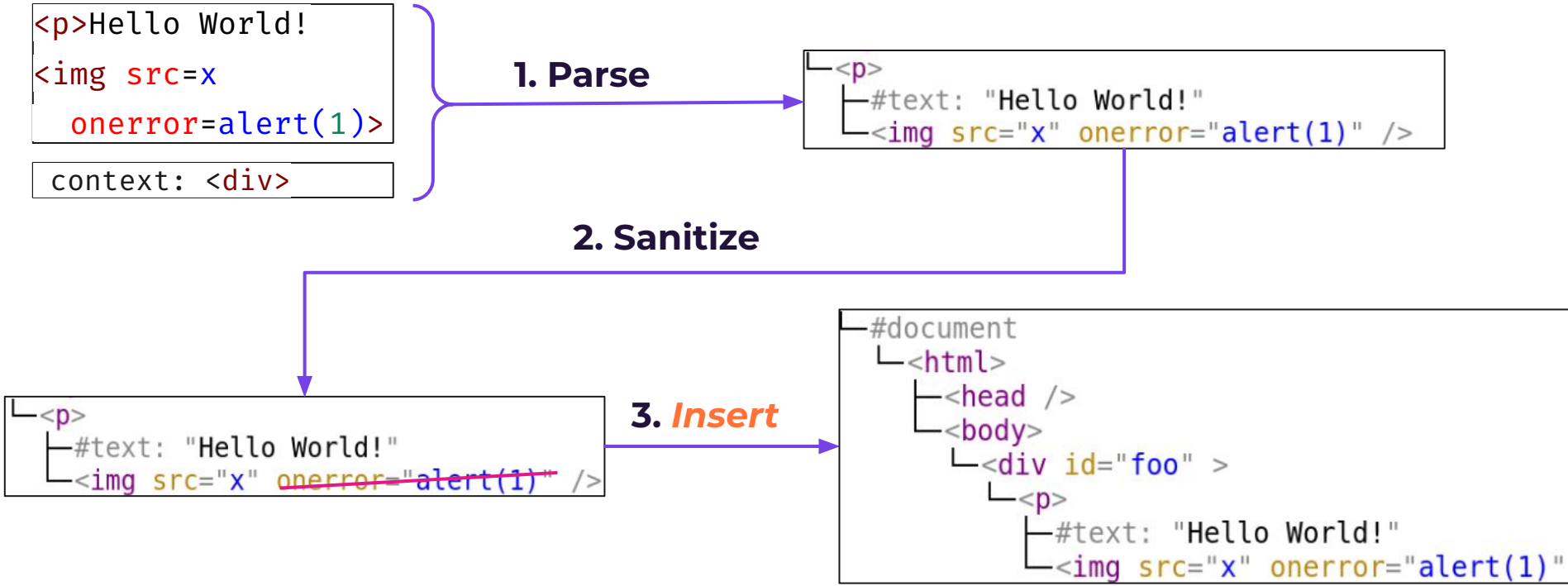
What do we want?



What do we want?



What do we want?



```
div.setHTML(
  evil,
  { sanitizer: mySanitizer }
)
```

`div.setHTML(evil)`

`"div.innerHTML = evil"`, but without XSS

Security Considerations for `.setHTML()`

But what about...

Server-Side Reflected and Stored XSS

DOM clobbering

XSS with Script gadgets

Mutated XSS

Server-Side XSS

The Sanitizer API is just for DOM-based XSS.

DOM clobbering

```
<form id=foo>
```

DOM clobbering

```
<form id=foo>
```

```
>> window.foo
<- ▶ <form name="foo">
>> document.foo
<- ▶ <form name="foo">
```

DOM Clobbering

e.g. Visibility

Document.hidden

The `Document.hidden` read-only property returns a Boolean value indicating if the page is considered hidden or not.

```
>> document.hidden
← false
```

DOM clobbering

```
<form id=hidden>
```

DOM clobbering

```
<form id=hidden>
```

```
>> document.hidden
<- ▶ <form name="hidden">
```

https://portswigger.net/web-security/dom-based/dom-clobbering

DOM clobbering

You can configure the sanitizer to disallow e.g., `name` & `id` attributes.

However, the default Sanitizer config only prevents XSS.

DOM clobbering

You could configure the sanitizer to disallow e.g., `name` & `id` attributes.

```
new Sanitizer({
  dropAttributes: [
    {name: "id", elements: "*"},
    {name: "name", elements: "*"},
]});
```

XSS with Script gadgets

```
<button data-html="injection here"
        data-html-enabled="true">
</button>
```

[Credit: https://research.google/pubs/pub46450/]

XSS with Script gadgets

You need to configure the Sanitizer according to your framework, by disallowing e.g., `data-` or `role` attributes.

The default Sanitizer config can not prevent these attacks.

mXSS

```
<svg></p>
<style>
<a id="</style><img src=1
onerror=alert(1)>">
```

mXSS

```
<svg></p>
<style>
<a id="</style><img src=1
onerror=alert(1)>">
```

```
<svg></svg>
<p></p>
<style><a id="</style>
<img src="1" onerror="alert(1)">
">
```

[Credit: https://research.securitum.com/dompurify-bypass-using-mxss/]

mXSS

The Sanitizer offers help against mXSS.

Parse at your own peril.

Improving with Feedback

We’re still not done here. Gimme moar bugs.

Bounties

1. Enable the Sanitizer

   Go to *about:config*. Toggle *dom.security.sanitizer.enabled*

   about://flags#sanitizer-api or "*Experimental Web Platform Features*"

2. Go to empty web page and open Developer Tools

3. `document.body.setHTML(evil)`

4. Profit

Discussion

https://wicg.github.io/sanitizer-api/

Bounties

1. Enable the Sanitizer

   Go to *about:config*. Toggle *dom.security.sanitizer.enabled*

   about://flags#sanitizer-api or "*Experimental Web Platform Features*"

2. Go to empty web page and open Developer Tools

3. `document.body.setHTML(evil)`

4. Profit

XSS is a browser bug

and the browser will fix it.

Thank you

Questions & Comments

- Matrix
  - @fbraun:mozilla.org
- Fediverse
  - @freddy@security.plumbing
- E-Mail
  - freddyb@mozilla.com