Developer's Guide to Cross Site Scripting

OWASP New Zealand Day 2017

whoami

Felix Shi (@comradepara)

- A security guy at Xero

- Infosec
- Running
- Cartography

Disclaimer: Something about my own opinions does not reflect those of my employer.

Disclaimer

Disclaimer: This is a primer to Cross Site Scripting (XSS), it is by no means an exhaustive list.

Please consult your local security team or physician if you think you are suffering from XSS.

Presentation Overview

1. Background

- Fundamentals
- What is XSS
- Why should you care
- Why is it still an issue
- Exploitation theory

2. Demo

- Exploitation practice
- Prevention theory
- Prevention practice
  - Backend
  - Frontend
  - Content Security Policy
- Mitigation practice
  - Input validation
  - Cookie Flags

Background

What's in a modern web application?

- Stuff the browser uses
  - HTML, Javascript, CSS, pretty pictures
- Stuff the server uses
  - Ruby, Java, C#, Python etc.
- Persistent server side storage
  - SQL databases, file systems

HTML

- Has been around since forever
  - (Correction: Invented in the late 80s)
- The building block of the web
- Elements on the page are described using tags

Fundamentals

HTML Tags

- `<b> Hello I’m bold </b>`
- `<u> Underlined </u>`
- `<img src=’tower.jpg’ />`

HTML Tags

- <b> Hello I’m bold </b>
- <u> Underlined </u>
- <img src='tower.jpg' />

Hello I’m bold

<u>Underlined</u>

Ways to include Javascript on a page

- `<script>console.log("Hello");</script>`
- `<script src="test.js" />`
- `<img src='hi.jpg' onload='alert(1)' />`

And many other ways!!!

What can you do with Javascript?

- Alter the look and functionality of the page
- Access private user data associated with the site
- Perform actions on the user's behalf

But I trust the webapps I use!

Let's talk about...

Cross Site Scripting!

What is Cross Site Scripting (XSS)?

What is Cross Site Scripting (XSS)?

Someone can get their own Javascript to run in the context of your site

Why should I care?

¯_(ツ)_/¯

Who does it affect?

How could it affect the user?

- The user's browser executes the malicious Javascript
- Alter the look and functionality of the page
- Access private user data associated with the site
- Perform actions on the user's behalf

¯\_(ツ)_/¯

Who does it affect?

How could it affect your company?

- Loss of **trust**
  - Bad PR
- Fixing technical debt is expensive
  - Which leads to angry product owners
  - Anger leads to hate, something… dark side
- Regulation / Compliance issues
  - Some certs require a clean pentest report

Why is it still an issue?

Why is it still an issue?

Because handling user defined data is **hard**

Exploitation Time!!!

XSS Exploitation Theory

- Identify the entry points of user defined data.
- Identify how the above data gets used on the page.
- The goal of XSS is to get the browser to execute user defined scripts.

XSS Exploitation Theory

- Identify the entry points of user defined data.
- Identify how the above data gets used on the page.
- The goal of XSS is to get the browser to execute user defined scripts.

Types of Cross Site Scripting - **Reflected**

Example URL

```
http://trustedsite/search.php?q=<script>alert(1);</script>
```

Page source returned to the victim

```
<html>...<div>
        <script>alert(1);</script>
</div>...</html>
```

Exploitation Vector:

Social Engineering, an attacker crafts a URL and gets people to click on it.

Script Entry Point

- Various places, all ending up in persistent storage.
  - For example: Entries in a **guestbook**

Exploitation Vector

- User just needs to visit page that renders the stored script.

- More dangerous than reflected XSS.
  - Can be prepared in advance
  - Can affect multiple users

Types of Cross Site Scripting - **DOM Based**

Example user data

```
http://trustedsite/search.php?q=<script>alert(1);</script>
```

Page source excerpt

```
...<script>
        document.write(document.URL.indexOf("q=")+2);
</script>..
```

Note that the XSS script **does not** appear in the source code.

Demo Time!

:D

Defence

Prevention Theory

- XSS issues are introduced when user supplied Javascript snippets are executed by the browser
- Faulty handling of user provided data

Defence

- Multiple user defined strings were rendered on the page:
  - The title URL parameter
  - Username field
  - Message field

Defence

URL:

```
http://url/entries?title=<script>alert(1);</script>
```

HTML Output:

```
<h1>
        Thank you for signing my
        <script>alert(1);</script>
</h1>
```

Defence

- ~~Don't allow user input~~
  - Not possible IRL :(
- Ensure that user provided data is validated when appropriate
- Ensure that user provided data is properly encoded/escaped on output

What is Encoding ?????

Defence

HTML Encoding is a technique that converts potentially unsafe characters into their encoded form.

Character	HTML Encoded
<	&lt;
>	&gt;
&	&amp;

Defence - Encoding

Input:

```
<script>
    alert(1);
</script>
```

HTML Encoded Output:

```
&lt;script&gt;
    alert(1);
&lt;/script&gt;
```

Defence - Encoding

Input:

```
<script>
    alert(1);
</script>
```

HTML Encoded Output:

```
&lt;script&gt;
    alert(1);
&lt;/script&gt;
```

User sees:

<script>alert(1);</script>

Defence - Encoding

Input:

```
<script>
    alert(1);
</script>
```

HTML Encoded Output:

```
&lt;script&gt;
    alert(1);
&lt;/script&gt;
```

NO SCRIPT EXECUTION FOR YOU!!1 >:)

User sees:

```
<script>alert(1);</script>
```

HTML Encoding for Developers

Templates: Django, Flask, Rails v. > 3.0, Mustache for Node.JS

- Secure by default
  - Automatically HTML encodes user data

Opting out of HTML Encoding in Flask:

```
{{username | safe}}
```

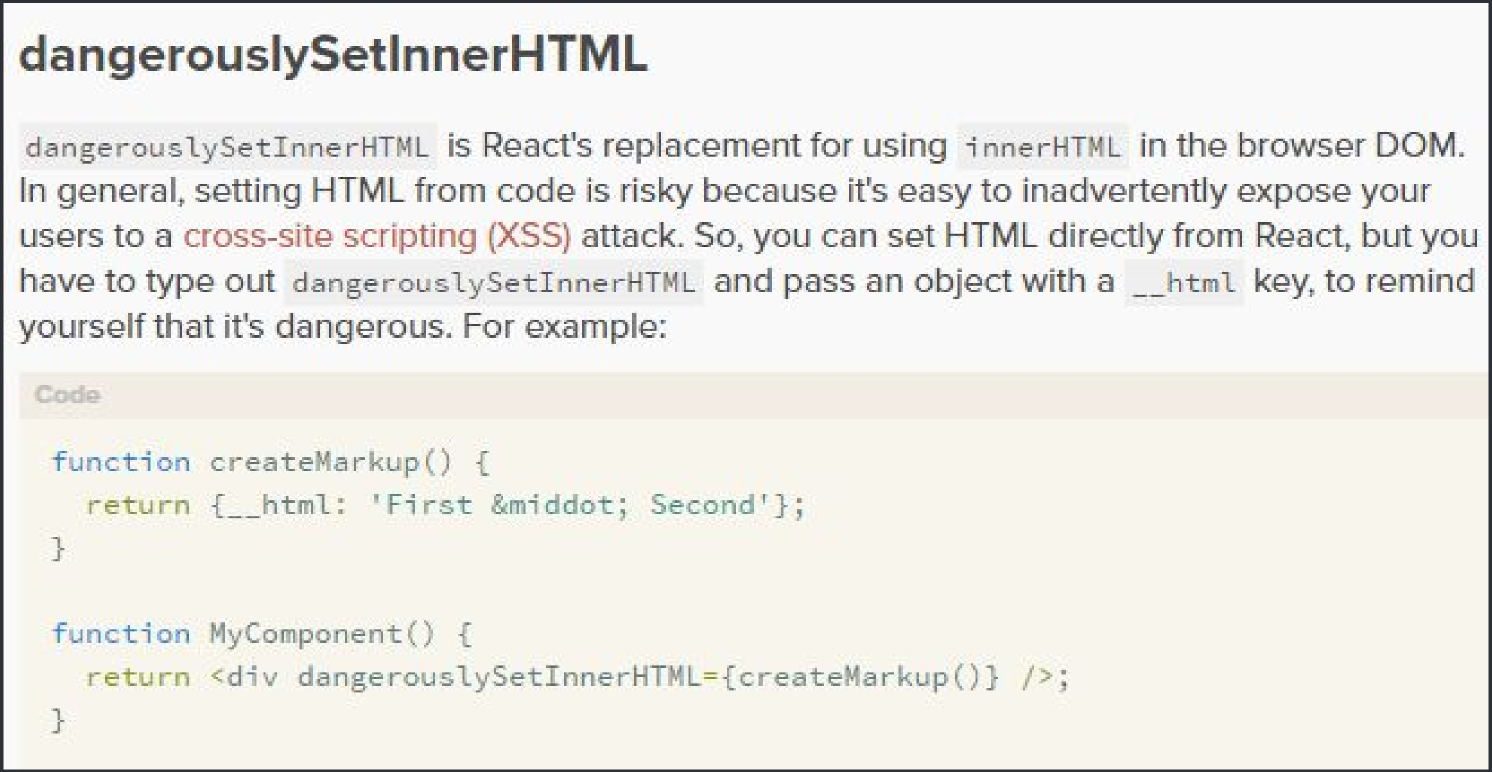
HTML Encoding for Developers

- Most modern front-end Javascript frameworks also HTML encode their output by default.
  - For example: Angular.js, React.js

Opting out of HTML Encoding in React.js...

Defence - Encoding (Frontend)

dangerouslySetInnerHTML


dangerouslySetInnerHTML

`dangerouslySetInnerHTML` is React's replacement for using `innerHTML` in the browser DOM. In general, setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack. So, you can set HTML directly from React, but you have to type out `dangerouslySetInnerHTML` and pass an object with a `__html` key, to remind yourself that it's dangerous. For example:

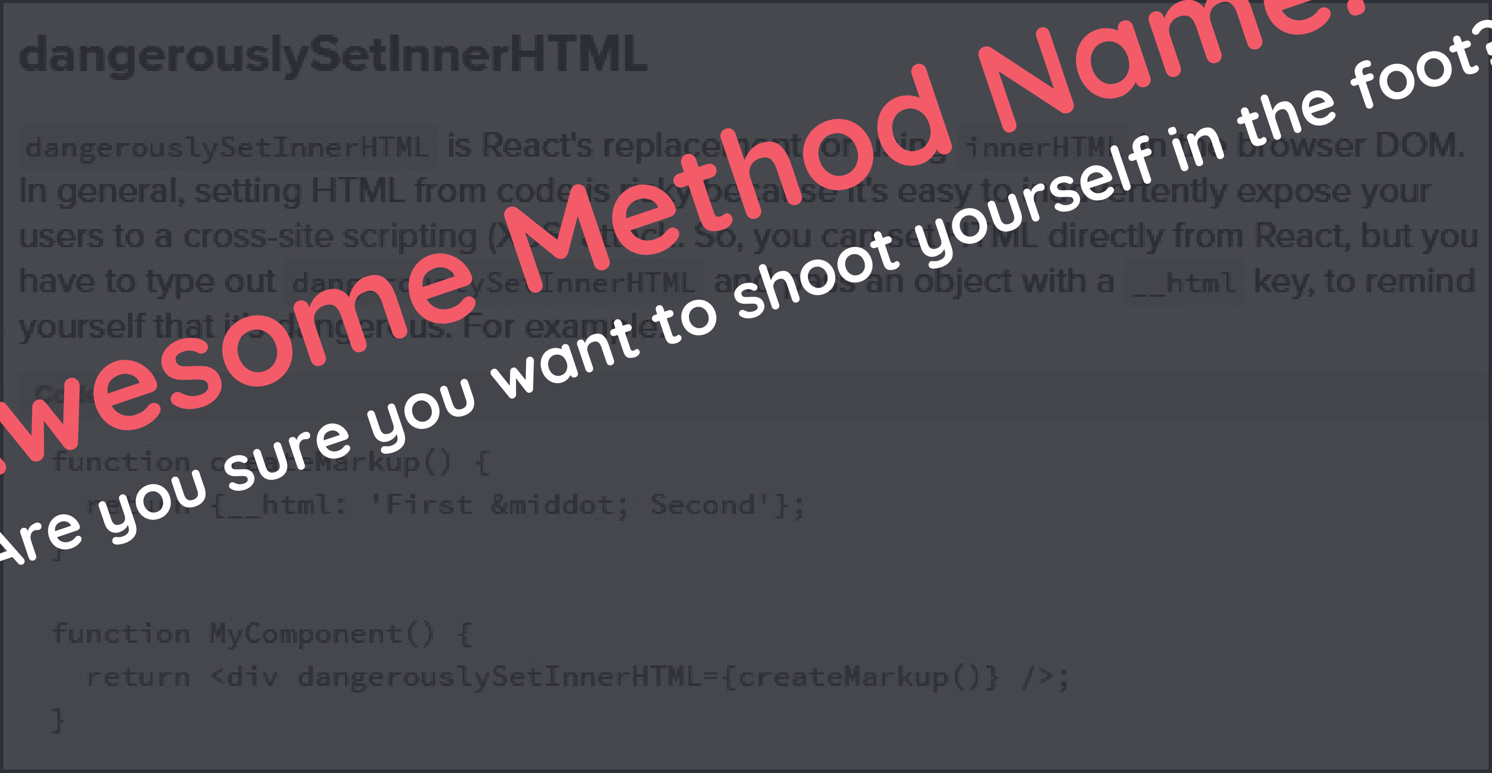
Code

```
function createMarkup() {
  return {__html: 'First &middot; Second'};
}

function MyComponent() {
  return <div dangerouslySetInnerHTML={createMarkup()} />;
}
```

Defence - Encoding (Frontend)

dangerouslySetInnerHTML



dangerouslySetInnerHTML

`dangerouslySetInnerHTML` is React's replacement for using `innerHTML` in the browser DOM. In general, setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack. So, you can set HTML directly from React, but you have to type out `dangerouslySetInnerHTML` and pass an object with a `__html` key, to remind yourself that it's dangerous. For example:

```
function createMarkup() {
  return {__html: 'First &middot; Second'};
}

function MyComponent() {
  return <div dangerouslySetInnerHTML={createMarkup()} />;
}
```

Awesome Method Name!

"Are you sure you want to shoot yourself in the foot?"

HTML Encoding for Developers

Still want to do encoding on the server-side manually?

- Use an established library!
    - .NET (If you are not using Razor)
        - System.Web.HttpUtility.HtmlEncode
    - Java
        - StringEscapeUtils.esapeHTML

Don't write your own encoding library

We
HTML Encoded
Everything!

It is

Demo Time

Again! :D

OH NOES! :(

Defence

- Another user defined data was found used the page:
  - Alternate text for the user's avatar

```
<img src='auto generated url'
alt='Username'/>
```

Defence

Username:

```
<script>alert(1);</script>
```

With HTML Encoding:

```
<img src = 'generated_url'
alt = '&lt;script&gt;alert(1);&lt;/script&gt;' />
```

Defence

Username:

```
' onload=alert(1) v='
```

With HTML Encoding:

```
<img src = 'generated_url'
alt = '' onload=alert(1) v='' />
```

Note: Not all HTML Encoder encodes the apostrophe character.

Defence

Username:

```
' onload=alert(1) v='
```

With HTML Encoding:

```
<img src = 'generated_url'
alt = '' onload=alert(1) v='' />
```

Note: Not all HTML Encoder encodes the apostrophe character.

Let's talk about Encoding

(Again)

Encoding Again

This time the user defined data was used inside a HTML attribute.

Other examples of user data in attributes:

```
<input type="text" value="user data" />
<img src="user data">
```

Another Encoding mechanism must be used in this scenario.

Attribute Encoding

Character	Attribute Encoded
'	&#39;
"	&quot;

Defence

Username:

```
' onload=alert(1) v='
```

With Attribute Encoding:

```
<img src = 'some auto generated url'
alt = '&#39; onload=alert(1) v=&#39;' />
```

Defence

Attribute Encoding for the Developers

If you are using templates

Make sure you wrap user input in quotes!

```
<img src="blegh" alt="{{user_input}}">
```

Attribute Encoding for the Developers

Use the appropriate attribute encoding method in your framework.

- Use an established library!
  - .NET
    - System.Web.HttpUtility.Html**Attribute**Encode
  - Java (OWASP Encoder)
    - org.owasp.encoder.Encode.forHTMLAttribute

Knowing
when to use
which encoding
is important!! :O

Context

HTML

```
<div>user input</div>
```

HTML Attribute

```
<input value="user input">
```

URL

```
http://mysite/index?title=user input
```

Context

Javascript Escaping

```
<script>var title = user input;</script>
```

Style / Cascading Style Sheet

```
background-image: user input;
```

And some others...

Context

Sometimes you need to use multiple encodings!

```
<script>
var title = ' ';alert(123); </script>
<script>alert(1);//';
</script>
```

Context

Sometimes you need to use multiple encodings!

```
<script>
    var title = ' ';alert(123);
</script>
<script>
    alert(1);//';
</script>
```

Context

Sometimes you need to use multiple encodings!

```
<script>
    var title = ' ';alert(123);
</script>
<script>
    alert(1);//';
</script>
```

More ways to prevent XSS

:D

Input Validation

- Should you allow special characters such as **<** and **>** in some fields?
- A **whitelist** approach is always preferred over blacklist
- Reject fields that have failed validation
- Ensure that input validation is used consistently across all points of input

Prevention - Input validation

Input Validation

Special mention for user defined URLs!

```
<a href='user input'>My site</a>
```

Javascript can be embedded by prefixing the link with **javascript:**

For example:

```
<a href='javascript:alert(1);'>Website</a>
```

Input Validation

Special mention for user defined URLs!

```
<a href='user input'>My site</a>
```

Validation Strategy:

- Fail the validation if it starts with Javascript:
- Validate that the user data is a valid URL
- (Optional) Check if URL is on a blacklist

Prevention - Cookie Flags

Cookie Security Flags

- Prevent your precious session cookies from being stolen by evil Javascript with the following flags.

- **HttpOnly**: Cookie is not accessible via Javascript
- **Secure**: Cookie can only be sent via HTTPS

Content Security Policy (CSP)

Go to this talk to listen to hear it from the pros:

So we broke all CSPs… You won't guess what happened next! (16:00, the same room you are in)

- ***Lukas*** *Weichselbaum &* ***Michele*** *Spagnuolo*

Content Security Policy (CSP)

Go to this talk to listen to hear it from the pros:

So we broke all CSPs… You won't guess what happened next! (16:00, the same room you are in)

- ***Lukas** Weichselbaum & **Michele** Spagnuolo*

Content Security Policy (CSP)

Go to this talk to listen to hear it from the pros:

So we broke all CSPs… You won't guess what happened next! (16:00, the same room you are in)

- ***Lukas*** *Weichselbaum* & ***Michele*** *Spagnuolo*

Links:

https://speakerdeck.com/mikispag/so-we-broke-all-csps-dot-dot-dot-you-wont-guess-what-happened-next-michele-spagnuolo-and-lukas-weichselbaum

https://deepsec.net/docs/Slides/2016/CSP_Is_Dead,_Long_Live_Strict_CSP!_Lukas_Weichselbaum.pdf

Now For the Takeaway Message

(You don't have to put up with me for much longer)

Takeaway

Developers Developers Developers

- Know where user data's used on the page
- Know the frameworks you are using
- Encode / Escape user data properly
- Validate input when appropriate
- Set cookie security flags
- Use Content Security Policy

Takeaway

Testers Testers Testers

- Take note of pages that contain user data
- Test by inserting script and see if they executed
- Look for XSS as a part of your quality assurance process
- Use a proxy:
  - ZAP, Burp, Charles, Fiddle
- Ask your security team for guidance
- Automate whenever possible

Misc.

Useful Links

More info on XSS

https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

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

https://www.google.com/about/appsecurity/learning/xss/

https://excess-xss.com/

Test Strings for the QAs

http://ha.ckers.org/xss.html

http://htmlpurifier.org/live/smoketests/xssAttacks.php

Content Security Policy (CSP)

https://developers.google.com/web/fundamentals/security/csp/

https://content-security-policy.com/

Useful Links

Proxies:

Burp (free edition): http://portswigger.net/burp/

ZAP: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Fiddler: http://www.telerik.com/fiddler

Charles: https://www.charlesproxy.com/

Exercises:

The XSS Game: https://xss-game.appspot.com/

Google Gruyere: https://google-gruyere.appspot.com/

XSS/SQLi Lab VM Image: https://pentesterlab.com/exercises/xss_and_mysql_file

BeEF when you really want to mess around with XSS:

Browser Exploitation Framework (BeEF): https://github.com/beefproject/beef

Slide theme from slidescarnival.com

Cheers

Cheers

and have an awesome day! :D