

UF Student InfoSec Team

September 8, 2015

#ufsit on Freenode

Announcements

Upcoming CTF: CSAW

CSAW CTF (Qualifiers)

- September 18 @ 6pm - September 20 @ 6pm

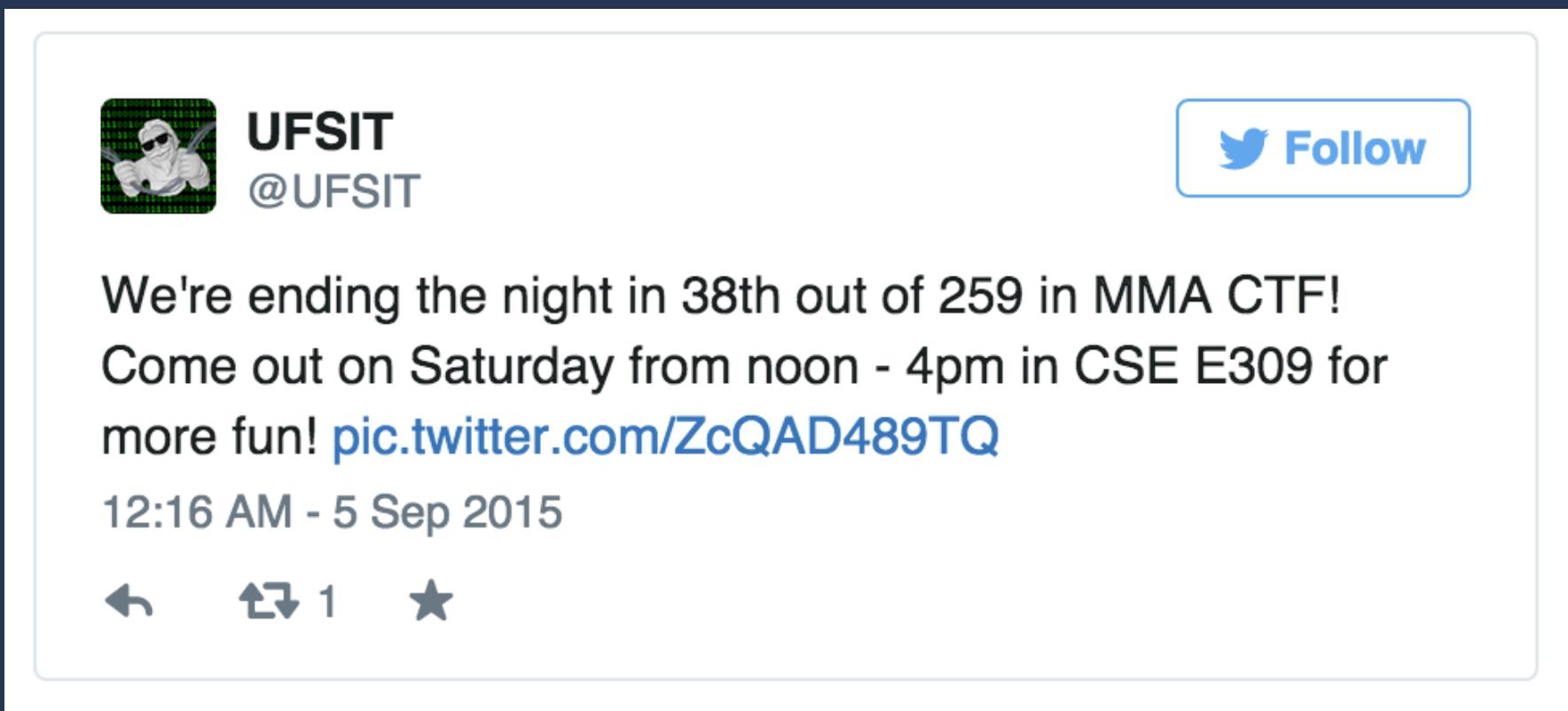
CSAW CTF (Qualifiers)

- September 18 @ 6pm - September 20 @ 6pm
- If we qualify, we get to send a team of undergrads to national CTF

MMA CTF Recap

MMA CTF Recap

- Lots of people there on Friday!
- Left Friday night in top 15%



We want your writeups!

Cross-site Scripting

(Also known as XSS)

Andrew Kerr | Sept 8, 2015
me@andrewjkerr.com

whoami

whoami

- Fifth year Software Engineering @ UF

whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs

whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs
- Full stack web developer



whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs
- Full stack web developer
- Former security intern at Tumblr



whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs
- Full stack web developer
- Former security intern at Tumblr
- Former intern at BlockScore

Cross-site Scripting

(Also known as XSS)

XSS

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

– OWASP

Ok... what does that mean?



andrewcore ▾



Title

◀ ▶ HTML

👁 Preview



#tags

Close

Post ▾



andrewcore ▾



Title

↔ HTML

👁 Preview



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

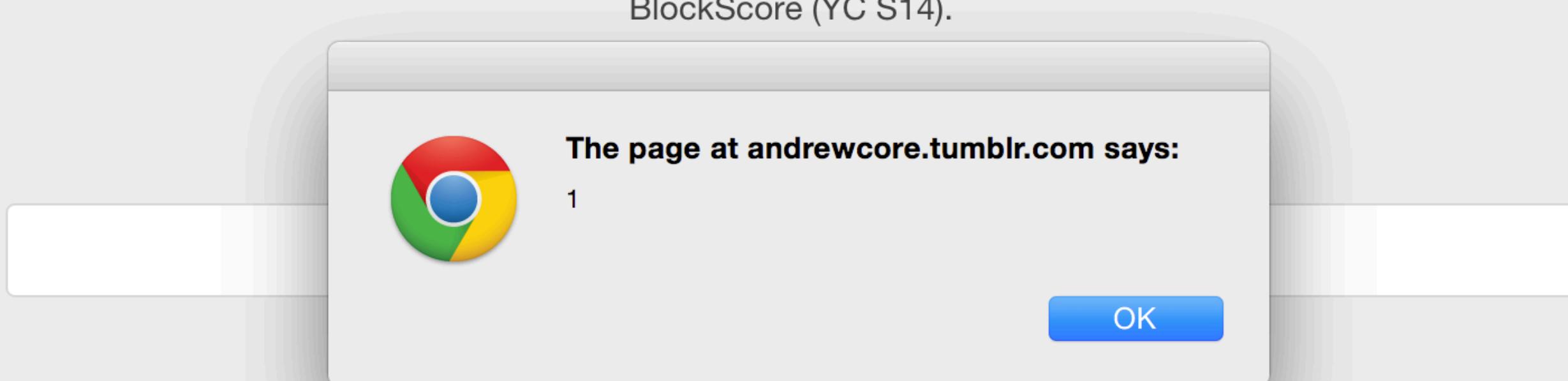
#tags

Close

Post ▾



Nerd. Currently skipping class at the University of Florida. Formerly engineering at Tumblr and BlockScore (YC S14).



Why does this work?

Why does this work?

- Browser is tricked into thinking the code is part of the site

Why does this work?

- Browser is tricked into thinking the code is part of the site
- Backend server does not sanitize input correctly

Why does this work?

- Browser is tricked into thinking the code is part of the site
- Backend server does not sanitize input correctly
- Poor client-side JavaScript executes given parameters

Why do it?

Why do it?

- Steal session cookies

Why do it?

- Steal session cookies
- Steal logins by defacing

Why do it?

- Steal session cookies
- Steal logins by defacing
- Exploit the browser/plugins

Why do it?

- Steal session cookies
- Steal logins by defacing
- Exploit the browser/plugins
- For the lulz



*andy

@derGeruhn



+ Follow

```
<script  
class="xss">$('.xss').parents().eq(1).find('a').e  
q(1).click(); $('[data-  
action=retweet]').click(); alert('XSS in  
Tweetdeck')</script> ❤
```

RETWEETS
71,317

FAVORITES
10,509



12:36 PM - 11 Jun 2014



...

xss">\$('.xss').parents().eq(1).find('a').click();\$('[data-retweet]').click();alert('XSS in TweetDeck')</script> ❤

/ORITES
0,509



Jun 2014



XSS | Andrew Kerr



Remember this?

- ❤ emoji broke XSS sanitization on TweetDeck

<xss">\$('.xss').parents().eq(1).find('a').click();\$('[data-retweet]').click();alert('XSS in TweetDeck')</script> ❤

/ORITES
0,509



Jun 2014



XSS | Andrew Kerr



Remember this?

- ❤ emoji broke XSS sanitization on TweetDeck
- Auto-magically retweeted itself 70,000+ times

<xss">\$('.xss').parents().eq(1).find('a').click();\$('[data-retweet]').click();alert('XSS in TweetDeck')</script> ❤

/ORITES
0,509



Jun 2014



XSS | Andrew Kerr



Remember this?

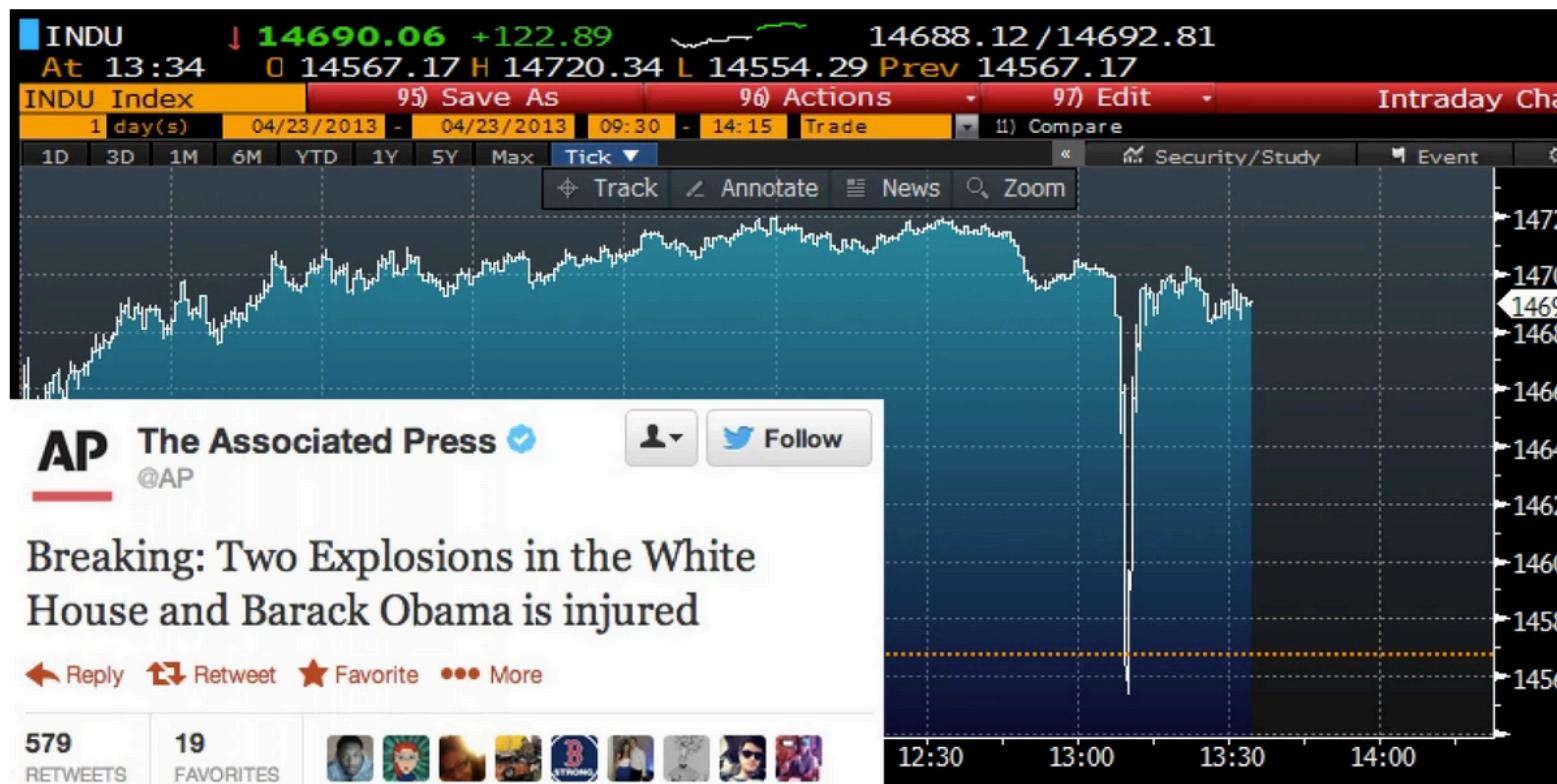
- ❤ emoji broke XSS sanitization on TweetDeck
- Auto-magically retweeted itself 70,000+ times
- Good thing it wasn't malicious!

Ok, but it's Twitter... why does it matter?

Syrian hackers claim AP hack that tipped stock market by \$136 billion. Is it terrorism?

A   0

By Max Fisher April 23, 2013 



This chart shows the Dow Jones Industrial Average during Tuesday afternoon's drop, caused by a fake A.P. tweet, inset at left.

XSS | Andrew Kerr

Most Read World

- 1 Pope calls on Europe's Catholics to take in refugees 

- 2 The Arab world's wealthiest nations are doing next to nothing for Syria's refugees 

- 3 An American family saved their son from joining the Islamic State. Now he might go to prison. 

- 4 The push and pull of China's orbit

- 5 Germans are obsessed with the U.S. But they'll deny it 

XSS Payloads

XSS Payloads

- A **TON** of possible XSS payloads

XSS Payloads

- A **TON** of possible XSS payloads
- <script>alert(1)</script>
-
- Click me!
- and more!

Types of XSS

Types of XSS

1. Reflected

Types of XSS

1. Reflected
2. Stored

Types of XSS

1. Reflected
2. Stored
3. DOM-based

Reflected XSS

Reflected XSS

- Ability to inject code and have the server return it back, unsanitized
 - Not stored on the server/in a database!

Reflected XSS

- Ability to inject code and have the server return it back, unsanitized
 - Not stored on the server/in a database!
- Normally hidden in the URL
 - Don't click on random links!

Reflected XSS

- Ability to inject code and have the server return it back, unsanitized
 - Not stored on the server/in a database!
- Normally hidden in the URL
 - Don't click on random links!
- Example: search forms showing input on results page after submission

Reflected XSS Vulnerable Code Example

```
// www.site.com/search.php?q=search+query
$search_query = $_GET['q'];
echo '<h1>Search results for: ' . $search_query . '</h1>;
```

**www.site.com/search.php?
q=<script>alert(1)</script>**

Search results for:



The page at <https://fiddle.jshell.net> says:

1

OK

Reflected XSS Vulnerable Code Example

```
// www.site.com/search.php?q=search+query
$search_query = $_GET['q'];
echo '<h1>Search results for: ' . $search_query . '</h1>;
```

Q: What's wrong with this code?

Reflected XSS Vulnerable Code Example

```
// www.site.com/search.php?q=search+query
$search_query = $_GET['q'];
echo '<h1>Search results for: ' . $search_query . '</h1>;
```

Q: What's wrong with this code?

A: UNSANITIZED USER INPUT

Stored XSS

Stored XSS

- Ability to inject code and have the server store it and return it without sanitizing it in either case

Stored XSS

- Ability to inject code and have the server store it and return it without sanitizing it in either case
- HOLY CRAP THIS IS HORRIBLE
 - Only way for end user to protect themselves is to disable JS

Stored XSS

- Ability to inject code and have the server store it and return it without sanitizing it in either case
- HOLY CRAP THIS IS HORRIBLE
 - Only way for end user to protect themselves is to disable JS
- Example: form post storing XSS



*andy

@derGeruhn



+ Follow

```
<script  
class="xss">$('.xss').parents().eq(1).find('a').e  
q(1).click(); $('[data-  
action=retweet]').click(); alert('XSS in  
Tweetdeck')</script> ❤
```

RETWEETS
71,317

FAVORITES
10,509



12:36 PM - 11 Jun 2014



...

Samy MySpace worm

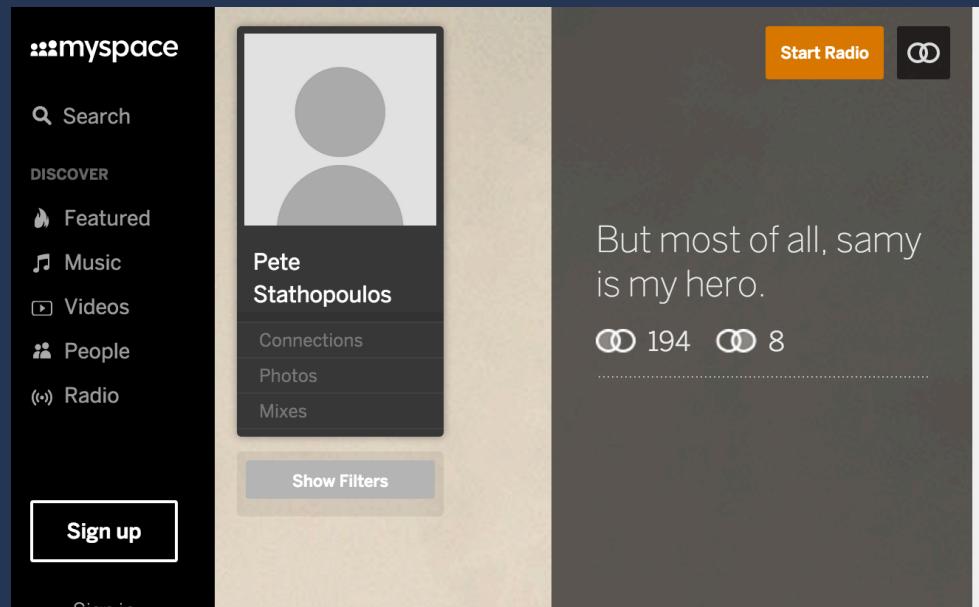
Samy MySpace worm

- Posted 'but most of all, samy is my hero' to victims



Samy MySpace worm

- Posted 'but most of all, samy is my hero' to victims
- Fastest spreading virus of all time
 - 1+ million runs in ~20hrs



Stored XSS Vulnerable Code Example

```
// Storing posts
$post = $_POST['post'];
$query = $mysql_conn->prepare("INSERT INTO posts VALUES ('" . $post . "'')");
$query->execute();

// Fetching and outputting posts
$query = $mysql_conn->prepare("SELECT * FROM posts");
$query->execute();
$query->bind_result($post);
while($query->fetch()) {
    echo '<p>' . $post . '</p>';
}
```

Stored XSS Vulnerable Code Example

```
// Storing posts
$post = $_POST['post'];
$query = $mysql_conn->prepare("INSERT INTO posts VALUES ('" . $post . "')");
$query->execute();

// Fetching and outputting posts
$query = $mysql_conn->prepare("SELECT * FROM posts");
$query->execute();
$query->bind_result($post);
while($query->fetch()) {
    echo '<p>' . $post . '</p>';
}
```

Q: What's the issue?

Stored XSS Vulnerable Code Example

```
// Storing posts
$post = $_POST['post'];
$query = $mysql_conn->prepare("INSERT INTO posts VALUES ('' . $post . '')");
$query->execute();

// Fetching and outputting posts
$query = $mysql_conn->prepare("SELECT * FROM posts");
$query->execute();
$query->bind_result($post);
while($query->fetch()) {
    echo '<p>' . $post . '</p>';
}
```

Q: What's the issue?

A: UNSANITIZED USER INPUT

DOM-based XSS

DOM-based XSS

- Similar to Reflected, *but* is not rendered from the server.

DOM-based XSS

- Similar to Reflected, *but* is not rendered from the server.
- Normally due to bad JavaScript code

DOM-based XSS

- Similar to Reflected, *but* is not rendered from the server.
- Normally due to bad JavaScript code
- Also crafted by a URL
 - Don't let users pass in JS via the URL!

DOM-based XSS Vulnerable Code Example

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

DOM-based XSS Vulnerable Code Example

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

Q: And, what's the issue here?

DOM-based XSS Vulnerable Code Example

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

Q: And, what's the issue here?

A: UNSANITIZED USER INPUT

DOM-based XSS Vulnerable Code Example

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

Q: How would we exploit this?

DOM-based XSS Vulnerable Code Example

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

Q: How would we exploit this?

A: Craft a URL like:

```
www.site.com/page.html?title=<img src='x'
onerror='alert(1)' />
```

Protecting Against XSS

Protecting Against XSS

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

Protecting Against XSS

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

- jQuery provides a `.html` AND `.text`.

Protecting Against XSS

```
// Pretend parse_get_params is implemented :)
var title = parse_get_params('title');
$('.page-header').html("<h1>" + title + "</h1>");
```

- jQuery provides a `.html` AND `.text`.
- But, what's the difference?

Let's look at the documentation!

Let's look at the documentation!

(Aka RTFM)

Protecting Against XSS

Set the text contents of the matched elements.

- .text()

Set the HTML contents of each element in the set of matched elements.

- .html()

Protecting Against XSS

Linkhello

Linkhello

```
<html>
  <head>
    <title>Test Page</title>
    <script type="text/javascript" src="jquery.min.js"></script>
    <script type="text/javascript">
      $(function(){
        $("#div1").html('<a href="example.html">Link</a><b>hello</b>');
        $("#div2").text('<a href="example.html">Link</a><b>hello</b>');
      });
    </script>
  </head>

  <body>
    <div id="div1"></div>
    <div id="div2"></div>

  </body>
</html>
```

Protecting Against XSS

1. Know your framework/library/language!

Protecting Against XSS

1. Know your framework/library/language!
2. SANITIZE!

Protecting Against XSS

1. Know your framework/library/language!
2. SANITIZE!
3. Whitelist, not blacklist

Protecting Against XSS

1. Know your framework/library/language!
2. SANITIZE!
3. Whitelist, not blacklist
4. Headers

Protecting Against XSS

- Or, ya know, read the NSA's recommendations.
- https://www.nsa.gov/ia/files/factsheets/xssiadfactsheetfinal_web.pdf

Protect Against Cross Site Scripting (XSS) Attacks

Cross Site Scripting (XSS) is a vulnerability in web applications that allows an attacker to inject HTML, typically including JavaScript code, into a web page. XSS results from the intermingling of server code and user

whitelisting, such as I recommended. Internet whitelisting through the [Restricted Security Zone](#)

But most importantly...

TEST YOUR APPLICATION

**JUST
DO IT!**



Bypassing Filters

- Wonderful cheatsheet by OWASP: <https://www.owasp.org/index.php/XSSFilterEvasionCheatSheet>

Bypassing Filters

- Wonderful cheatsheet by OWASP: <https://www.owasp.org/index.php/XSSFilterEvasionCheatSheet>
- Also, some guess work helps!

Bypassing Filters Vulnerable Code Example

```
$input = $_POST['input'];
$sanitized = str_replace('script', '', $input);
```

Bypassing Filters Vulnerable Code Example

```
$input = $_POST['input'];
$sanitized = str_replace('script', '', $input);
```

Q: How could we get by this?

Bypassing Filters Vulnerable Code Example

```
$input = $_POST['input'];
$sanitized = str_replace('script', '', $input);
```

Q: How could we get by this?

A: Think about it :)

Resources

Resources

- OWASP

Resources

- OWASP
- The Web Application Hackers Handbook

Resources

- OWASP
- The Web Application Hackers Handbook
- Mutillidae Practice Application

Ok, cool, onto challenges!

104.236.76.214

Go here

Challenges

- Server: 104.236.76.214
- Source: github.com/ufsit/xss-challenges
 - Try not to use this!
- Cheatsheet: https://www.owasp.org/index.php/XSS_Filter_Evasion_CheatSheet