In this post, we discuss about how to test Cross Site Scripting (XSS) vulnerability in web apps. If you don’t know what XSS is I recommend reading it [here](https://en.wikipedia.org/wiki/Cross-site_scripting) and [here](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)). Typically when we encounter an input field, we straightaway go and insert the most loved XSS payload <script>alert(‘XSS’)</script> to see if we get an alert box like this.

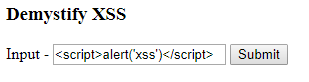


If that happens, great!

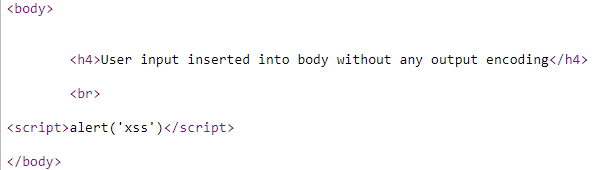
Just in case you do not get pop-up box, the application may still be vulnerable to XSS attack. Here is what we are going to do next. The first step is to figure out where the user input goes into the web page, I mean where in the HTML page the provided user input is inserted without any validation and output encoding.

Let us look at the following cases and observe where the user-input has gone into.

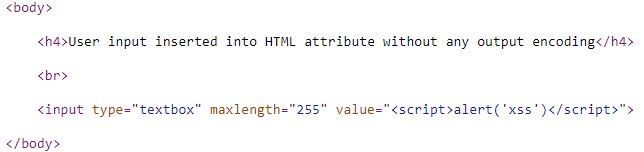
Web page with input field



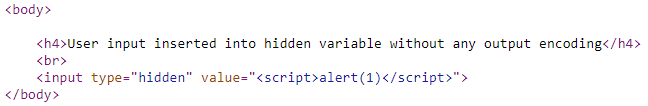
**1.** Context - body



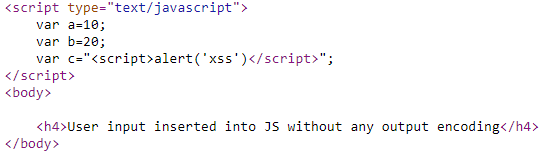
**2. Context - HTML attribute**



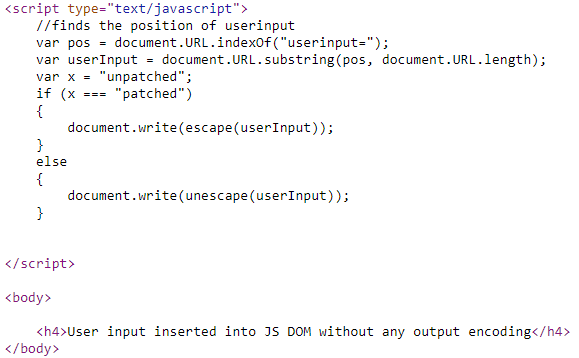
**3. Context - hidden variable**



**4. Context - javascript**



**5. Context - DOM**



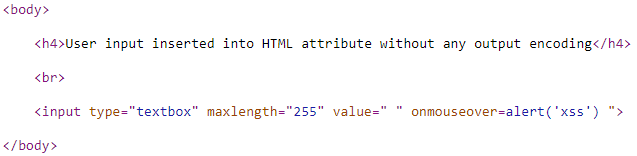
As you have rightly observed the user input has gone into various places or contexts i.e body, HTML attribute, hidden variable, javascript, DOM. The classical payload executes only in one case i.e. when the user input is inserted in the body.

In all other cases, the application is still vulnerable to XSS but the payload fails to execute. The reason is simple <script>alert(‘XSS’)</script>did not execute when inserted at other contexts like HTML attribute, DOM, javascript, hidden var.

In order to exploit other cases where the payload is inserted into different contexts we need to come up with context specific payload.

e.g. for HTML attribute context we can go with something like this

" onmouseover=alert('XSS')



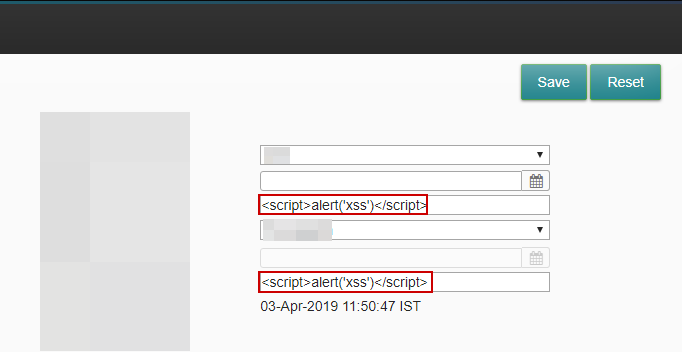
Similarly, we need to watch out where the user input is inserted and choose appropriate payload accordingly. We cannot just blindly go with <script>alert(‘XSS’)</script> and conclude whether an app is vulnerable to XSS.

**Testing AngularJS based apps for XSS**

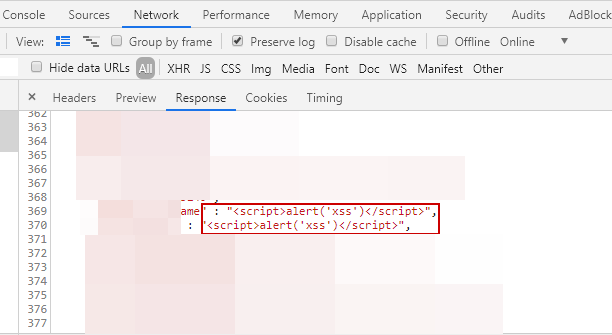
I’ve come across many angular apps where it’s difficult to find XSS vulnerability. The reason being before rendering the content angular does output encoding. This renders the payload useless and nothing more than a piece of plain text. With that said it doesn’t mean that angular apps are completely safe from XSS attacks. You can still find XSS through client side template injection. Check [here](https://hunter2.com/the-great-escape-avoiding-xss-in-angular-templates) and [here](https://portswigger.net/blog/xss-without-html-client-side-template-injection-with-angularjs).

We are not going to discuss template injection in this blog. We just take a quick look why our xss payload doesn’t execute when the frontend framework in angular JS.

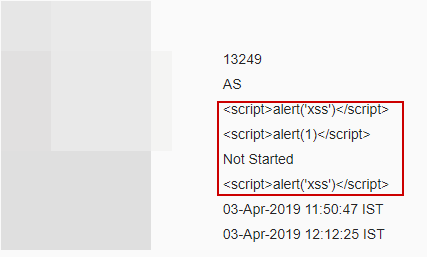
Let’s take this example, we inject our XSS payload.



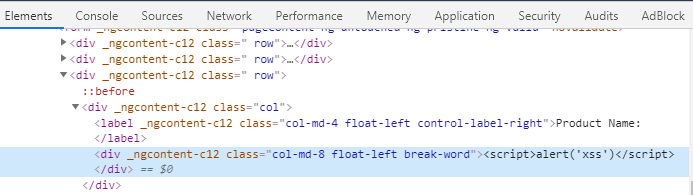
Server returns the payload without any output encoding.



But the payload doesn’t execute, it is rendered as plain text in the browser.

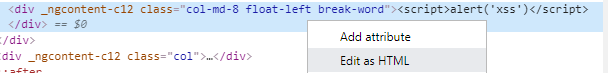


What’s wrong with the payload, lets inspect the element.

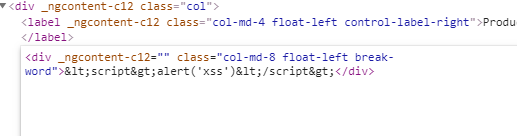


No clue here as you see your payload as it is without any output encoding. Confused why it did not execute?

Right click and go for edit as html



Now you can see clearly that the payload was output encoded and hence it did not execute



To learn more about angular Js security you can go for Troy Hunt’s course on [pluralsight](https://www.pluralsight.com/courses/angularjs-security-fundamentals).

I will upload this code to Github to experiment.

Feel free to say if you notice discrepancies if any.