Vulnerability Name:	JavaScript Attacks
Affected Vendor:	DVWA
Affected Product Name:	http://dvwa/vulnerabilities/javascript/
Product Official Website URL:	http://dvwa/login.php
Affected Component:	Phrase

Description: - JavaScript Attacks consist of a series of client-side attacks that are possible with the abuse of the client-side JavaScript used by the application. Most of the time, developer implements some JavaScript code in order to achieve certain security level at client side. However, an attacker can easily bypass these restrictions and perform malicious actions on the application.

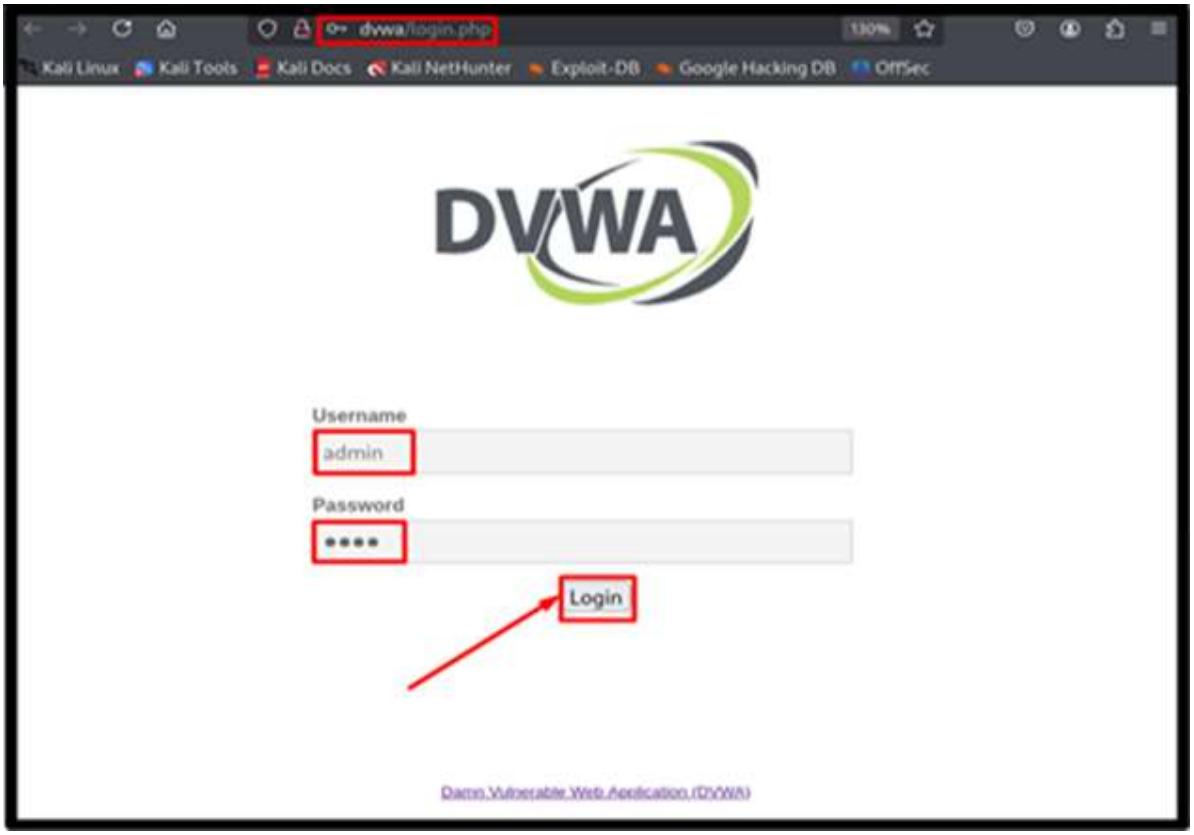
Root Cause: - The addition of JavaScript error detection allows teams to quickly begin to capture errors with minimal effort. Bugs are detected automatically and the stack trace along with other helpful attributes are sent to Railtown to give you details about the error.

Impact: - Data leakage, privacy violations, and business disruption
JavaScript attacks pose a significant risk to web security and can lead to serious consequences such as data leakage, privacy violations, and business disruption.
Mitigation: - Always use HTTPS, validate SSL/TLS certificates, implement CSP, secure cookies, and enforce HSTS to mitigate the risks of MitM attacks. Learn how to secure your JavaScript applications against Man-in-the-Middle (MitM) attacks with practical steps and real-world examples.

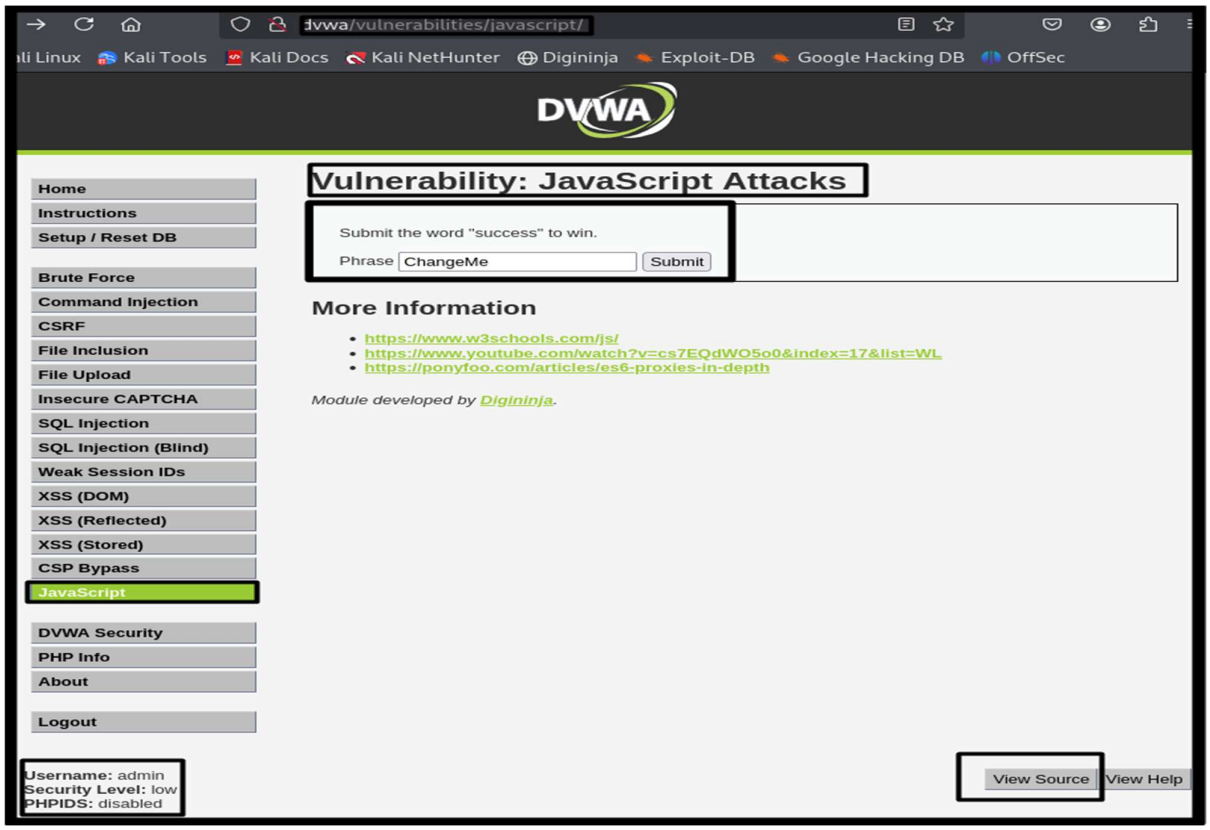
Remediation: - To remediate JavaScript attacks always validate and sanitize user input to prevent malicious scripts and implement CSP headers to restrict the execution of untrusted scripts. Encode or escape data before rendering it in the browser to prevent XSS and the use HTTPOnly and Secure flags on cookies to prevent access via JavaScript and never use eval() or similar functions to execute dynamic code and Implement proper authentication and authorization to prevent unauthorized access and keep all libraries and frameworks up-to-date to patch vulnerabilities.

Proof Of Concept

Step: -1 First navigate to http://dvwa/login.php and login with username and Password.



Step: -2 log in the home page of DVWA then click to the JavaScript Attacks Section.



SECURITY LEVEL(LOW)

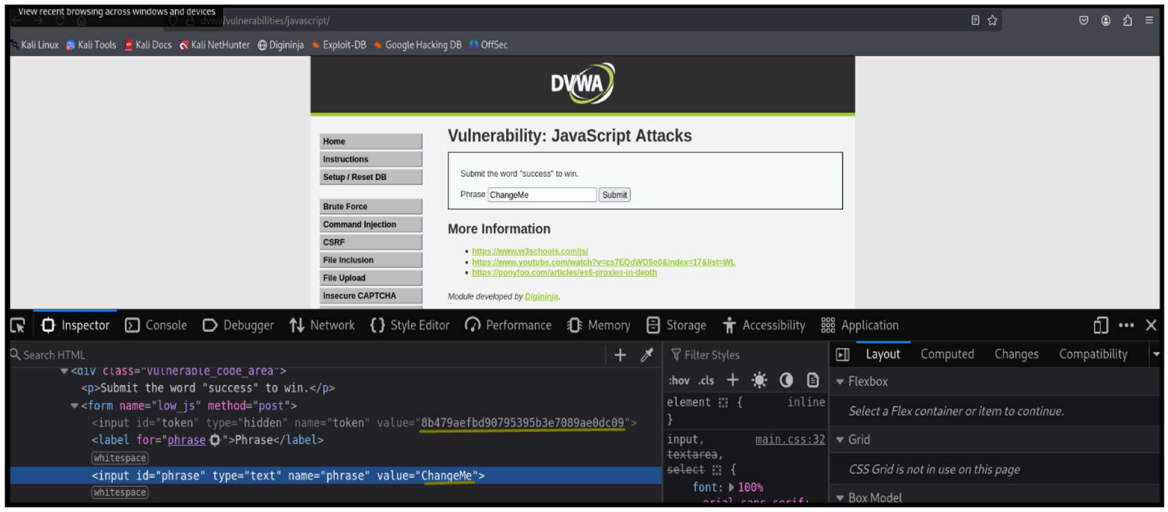
As we know we can first view the source code.we see that there is a function called generate_token() which takes our input and encodes it with ROT13 and then hashes it using MD5:

JavaScript Source

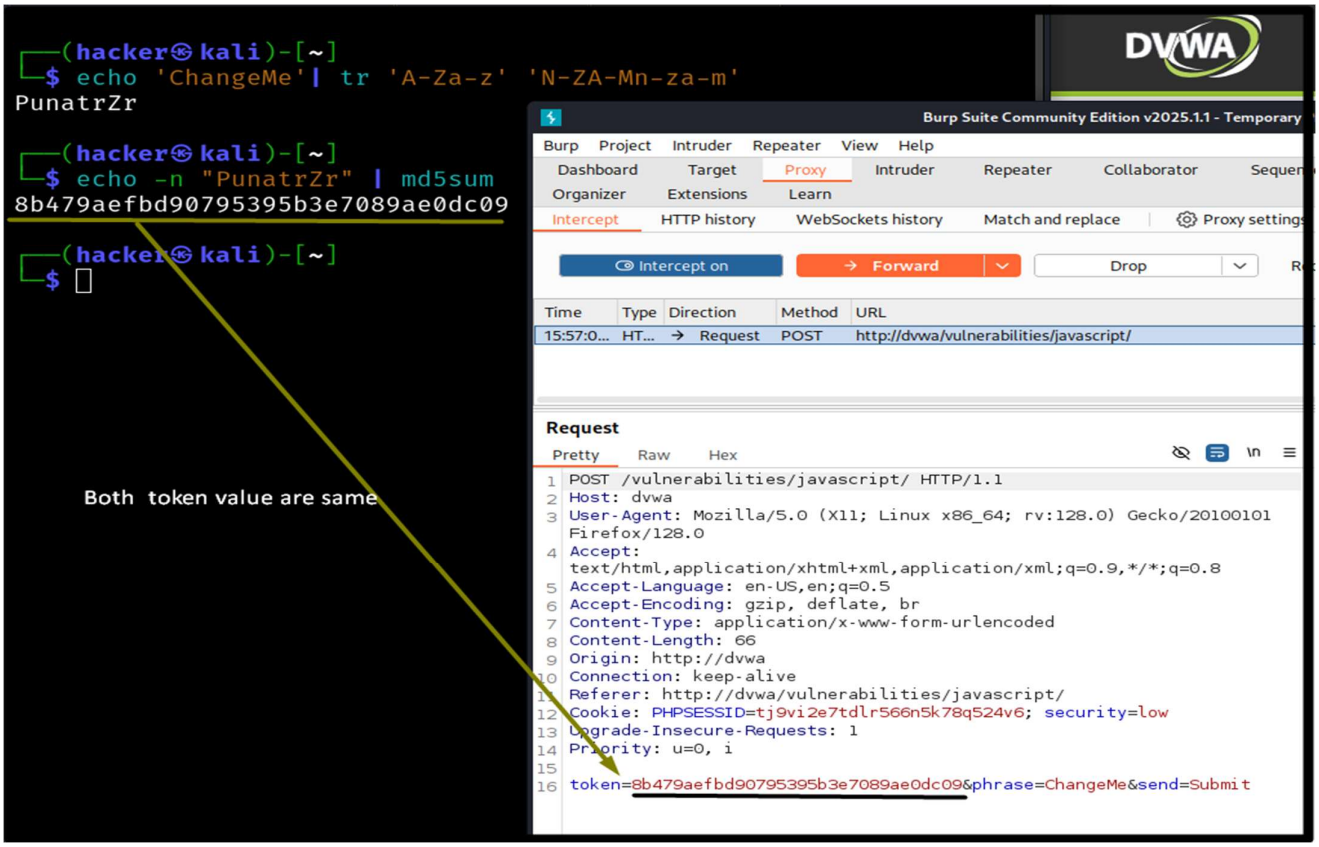
vulnerabilities/javascript/source/low.php

```
<?php
$page[ 'body' ] .= <<<EOF
<script>

/*
MD5 code from here
https://github.com/blueimp/JavaScript-MD5
*/

!function(n){"use strict";function t(n,t){var r=(65535&n)+(65535&t);return(n>>16)+(t>>16)+(r>>16)<<16|65535&r}function r(n,t){return n<<t|n>>>32-t}funct
=128<<r%32,n[14+(r+64>>>9<<4)]=r;var e,i,a,d,h,l=1732584193,g=-271733879,v=-1732584194,m=271733878;for(e=0;e<n.length;e+=16)i=l,a=g,d=v,h=m,g=f(g=f(g=f(
function a(n){var t,r="",e=32*n.length;for(t=0;t<e;t+=8)r+=String.fromCharCode(n[t>>5]>>>t%32&255);return r}function d(n){var t,r=[];for(r[(n.length>>2)

    function rot13(inp) {
        return inp.replace(/[a-zA-Z]/g,function(c){return String.fromCharCode((c<="Z"?90:122)>=(c=c.charCodeAt(0)+13)?c:c-26);});
    }

    function generate_token() {
        var phrase = document.getElementById("phrase").value;
        document.getElementById("token").value = md5(rot13(phrase));
    }

    generate_token();
</script>
EOF;
?>
```

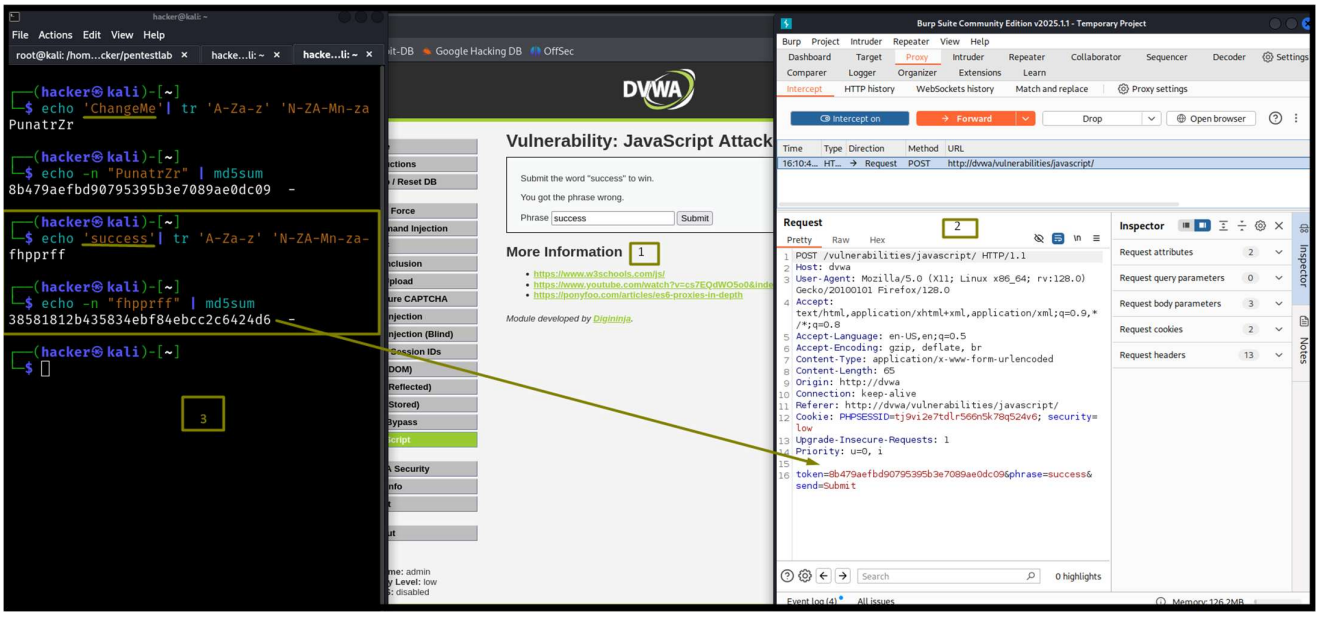
Step: -3 We can find the token mentioned by inspecting the page



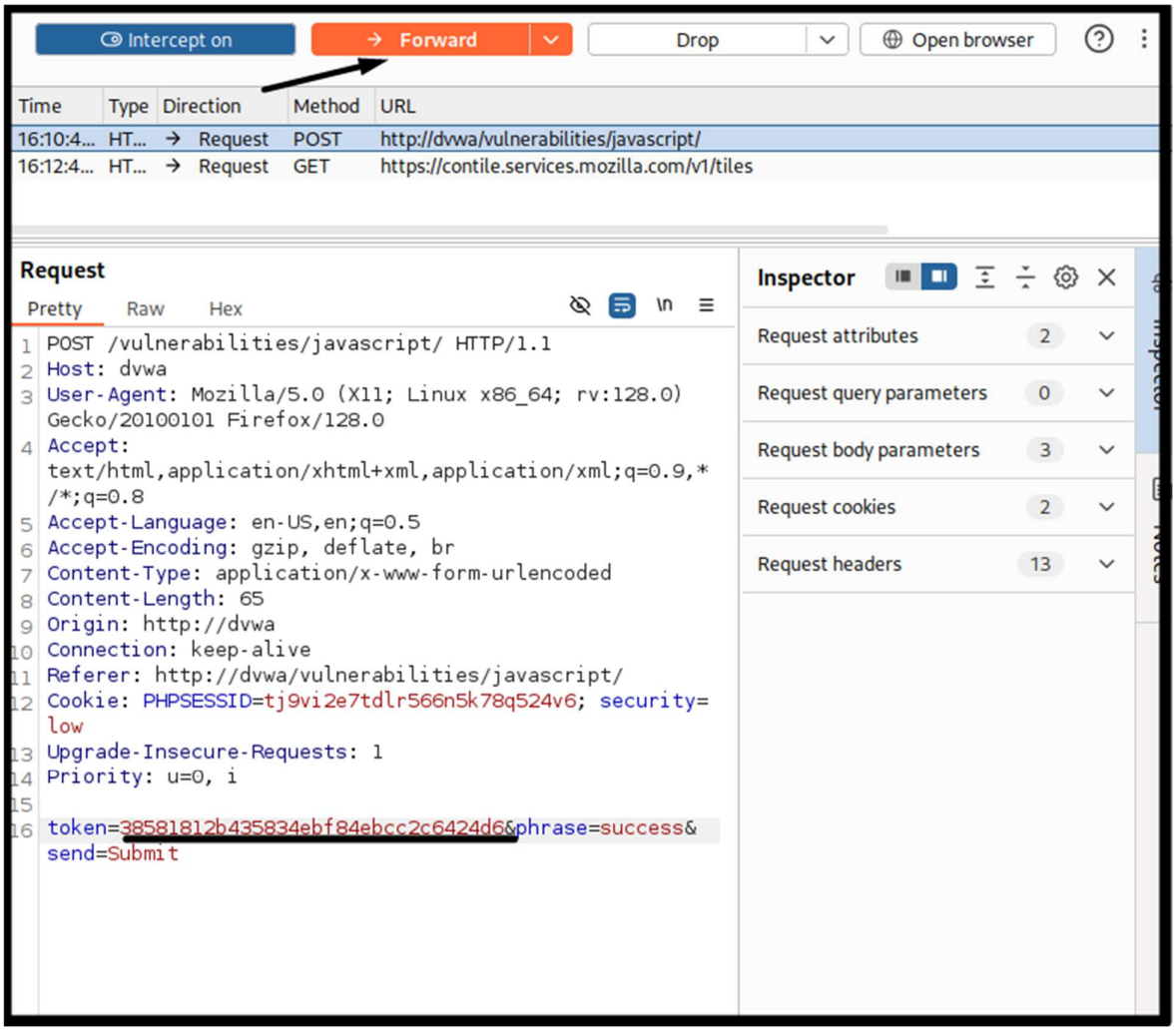
Step: -4 In this Step I am accepting the request in burp proxy. The we can see the both token values are same. similarly



Step: -5 "First, I capture the request in Burp Suite when I submit the form with a 'success' value. Then, I intercept the request, replace the 'success' token value with its ROT13 equivalent ('fhpprff'), and finally convert 'fhpprff' to its MD5 hash value."



Step: -6 Change the token value and forward the request



Step: -7 "Finally, submit the word 'success' to win, and the message 'Well done' will be displayed."

SECURITY LEVEL (MEDIUM)

As we know we can see the first view the source code.

the token is always XXeMegnahCXX. If the phrase is not success we get the error You got the phrase wrong. When the phrase is success we get the error Invalid token.

JavaScript Source

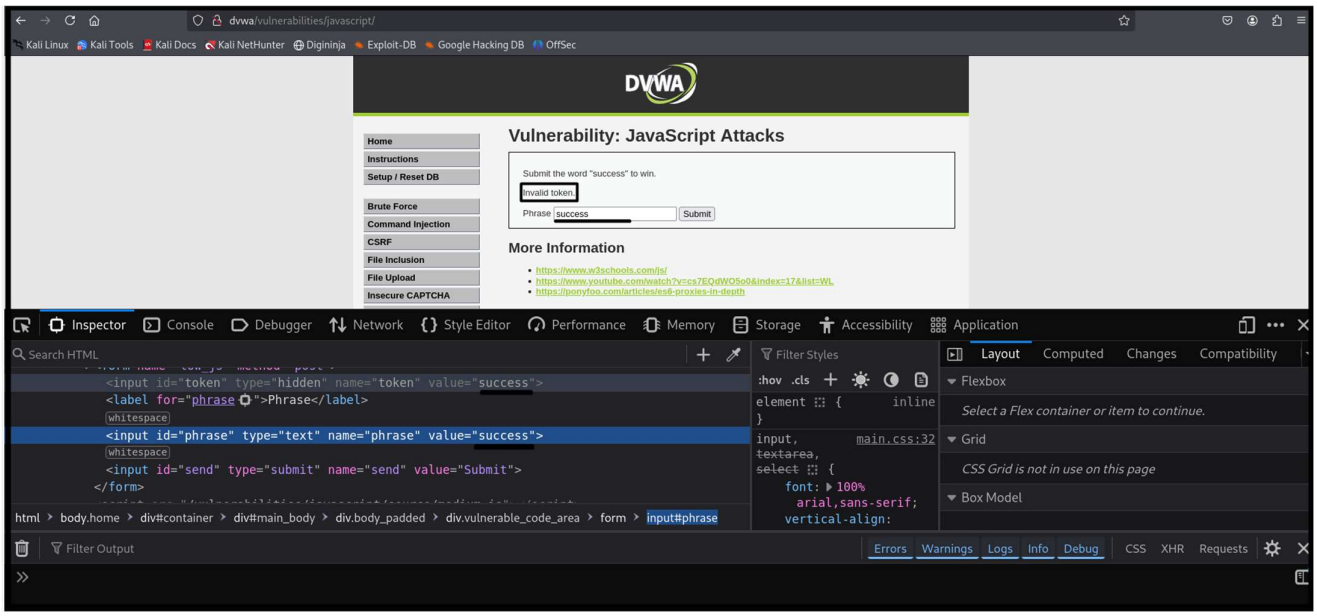
vulnerabilities/javascript/source/medium.php

```
<?php
$page[ 'body' ] .= <<<EOF
<script src="/vulnerabilities/javascript/source/medium.js"></script>
EOF;
?>
```

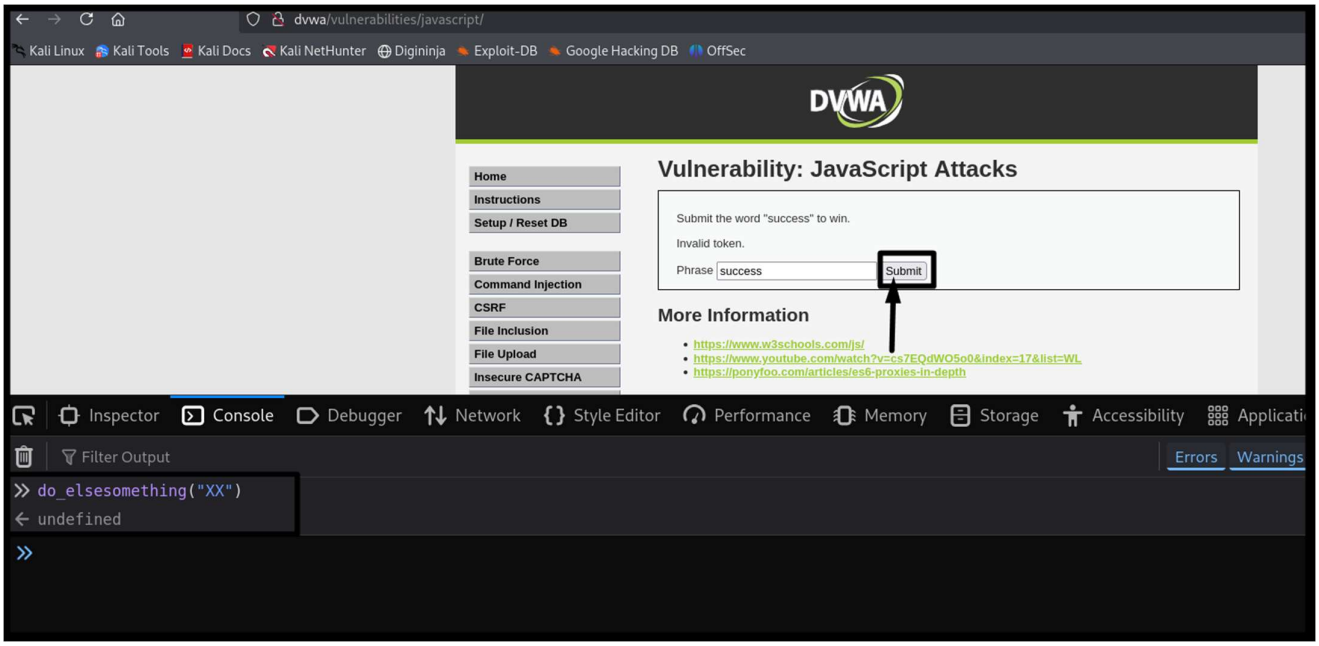
vulnerabilities/javascript/source/medium.js

```
function do_something(e){for(var t="",n=e.length-1;n>=0;n--)t+=e[n];return t}setTimeout(function()
{do_elsesomething("XX")},300);function do_elsesomething(e)
{document.getElementById("token").value=do_something(e+document.getElementById("phrase").value+"XX")}
```

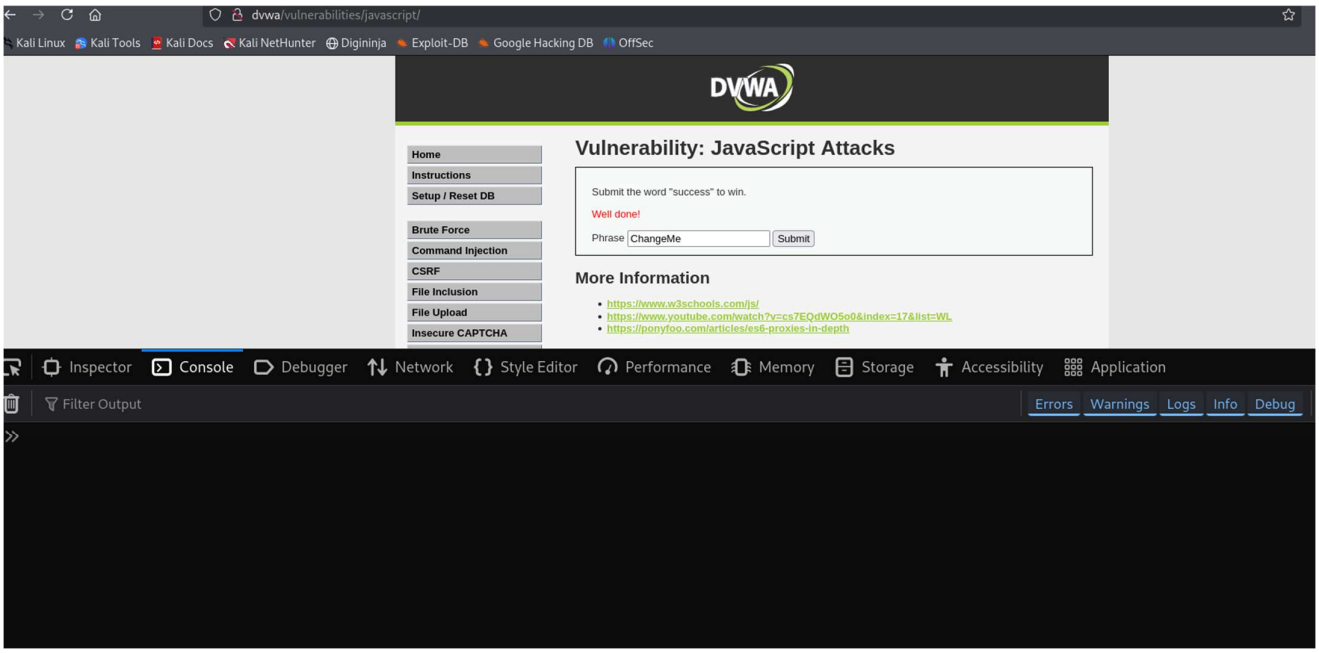
Step: -1 We can find the token mentioned by inspecting the page and both the values change token value and phrase value(success).



Step: -2 Go to console and put the value do_elsesomething(“XX”) and hit press enter and click to submit button

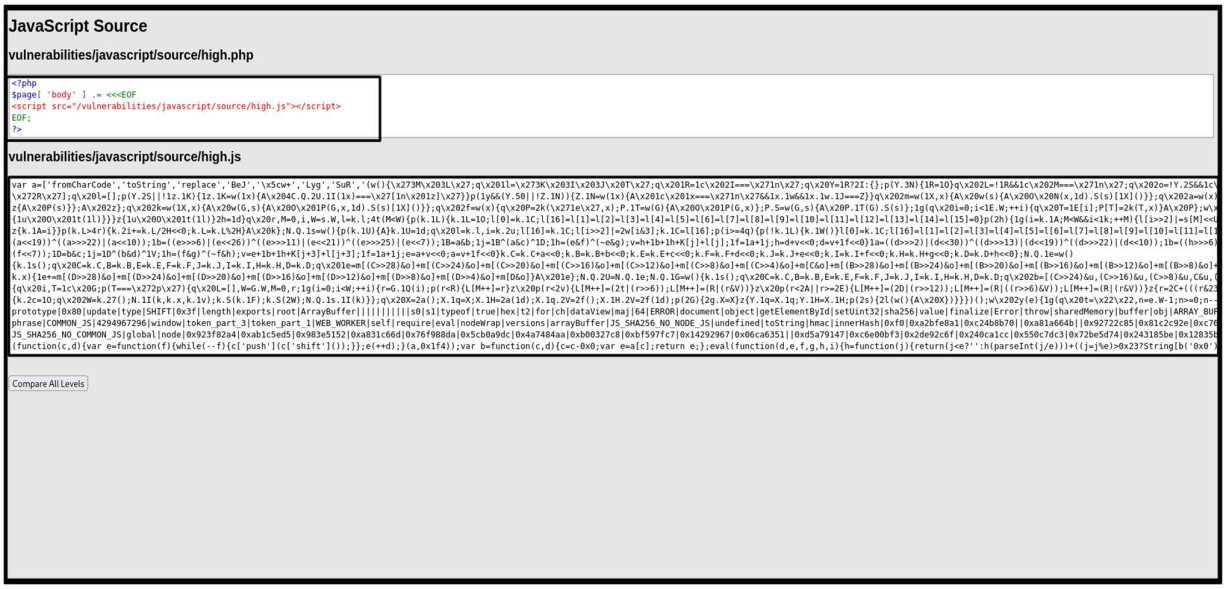


Step: -3 Finally, submit the word 'success' to win, and the message 'Well done' will be displayed.

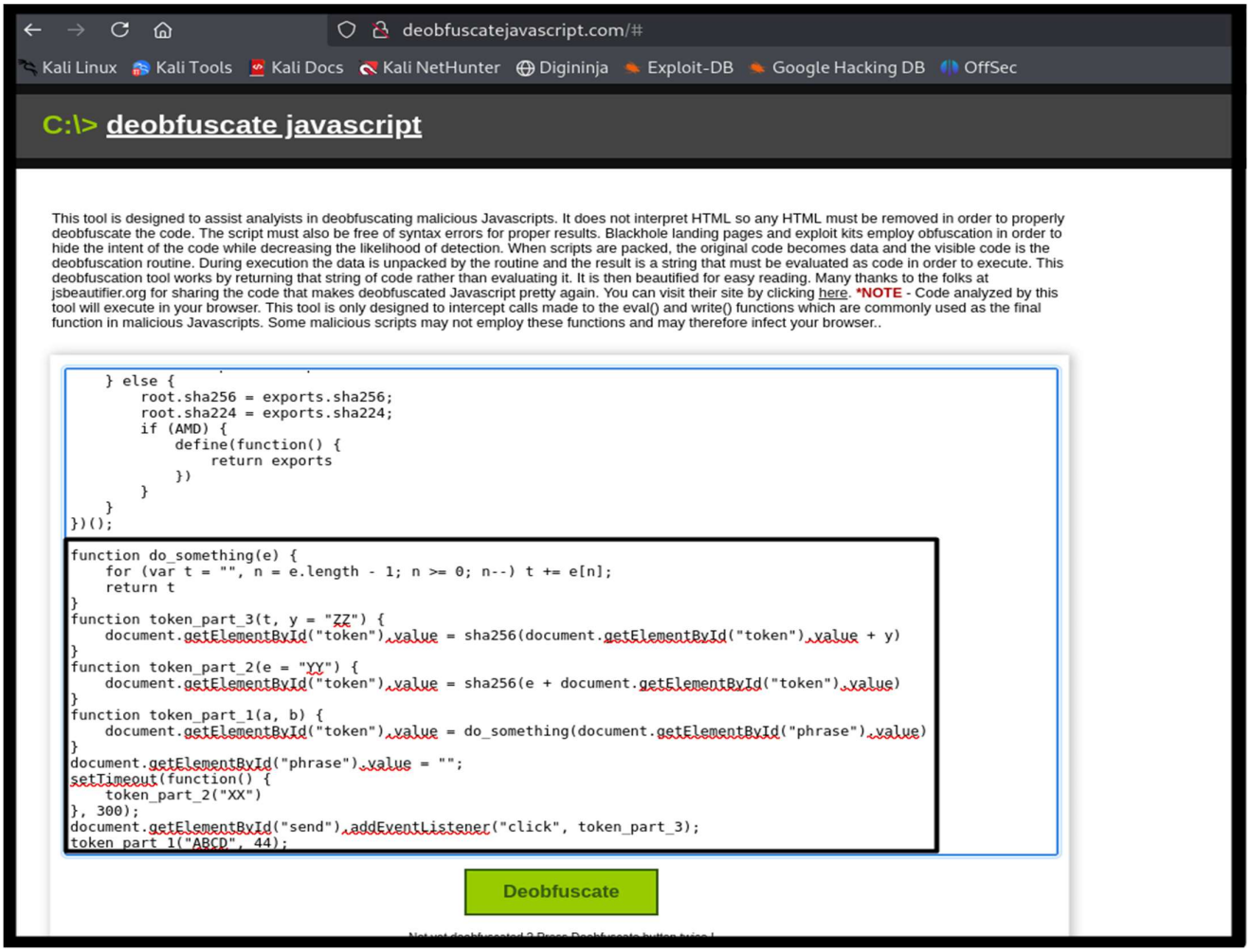


SECURITY LEVEL (HIGH)

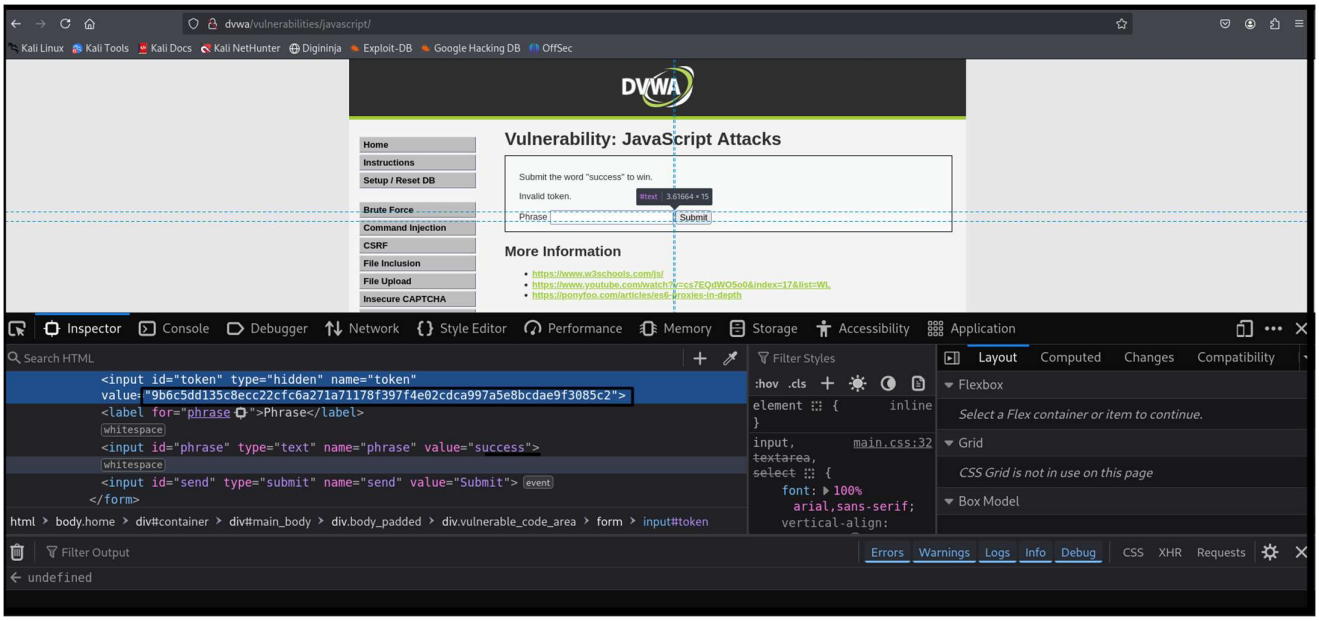
As we know we can see the view page source code. we see that there is a function called generate_token() which takes our input and encodes.



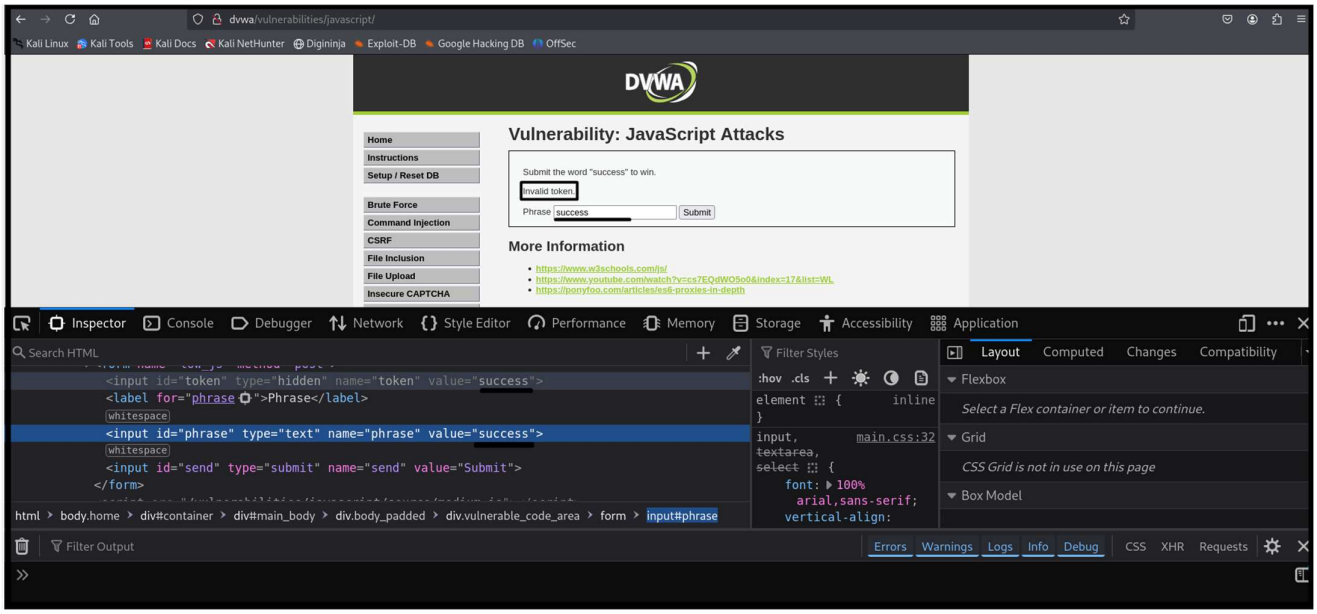
Deobfuscate.javascript we see that there is a function called generate_token() . but Now I understand this code.



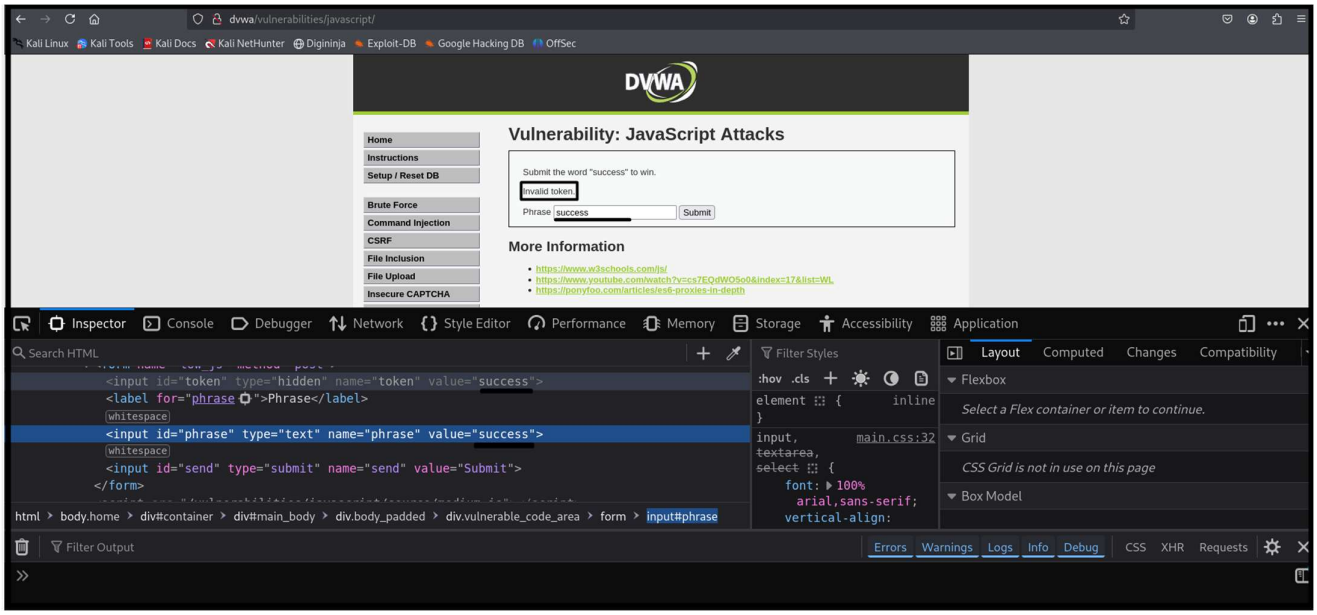
Step: -1 We can find the token mentioned by inspecting the page and both the values change token value and phrase value(success). **:-**



Step: - 2 In this Step both values are changes token value and phrase values to success.



Step: - 3 In this step fill the phrase name success.



Step: -4 In this step, select this file:

html

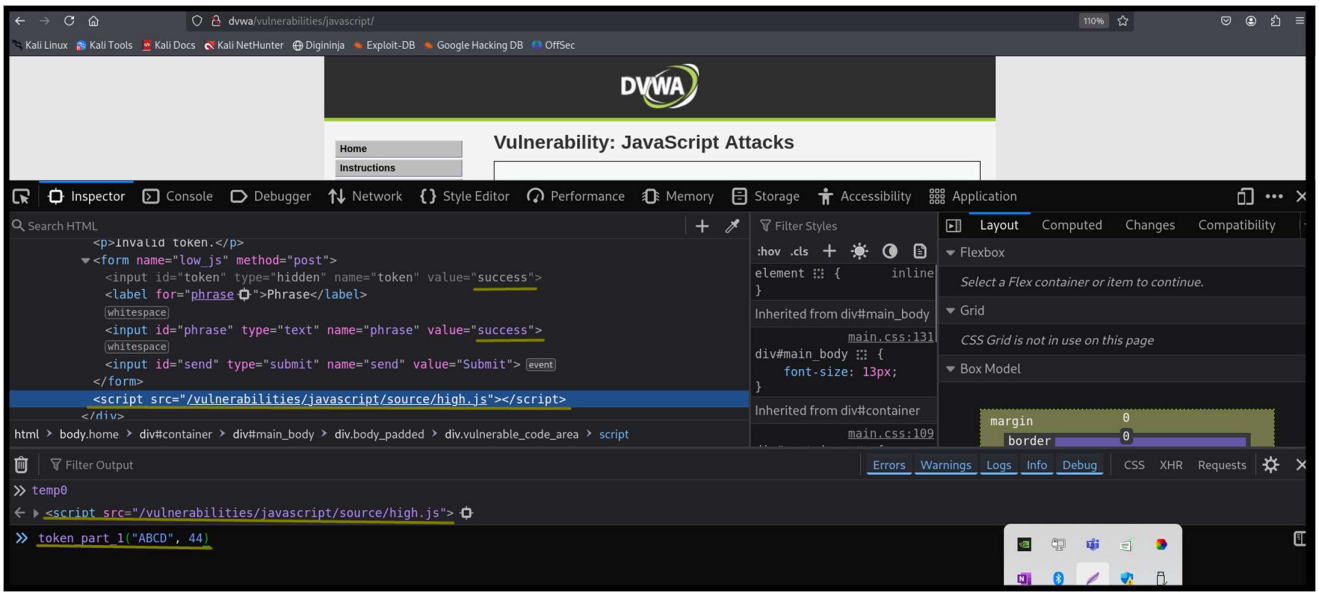
<script src="vulnerabilities/javascript/source/high.js"></script>

Then, open the browser console and enter the following command:

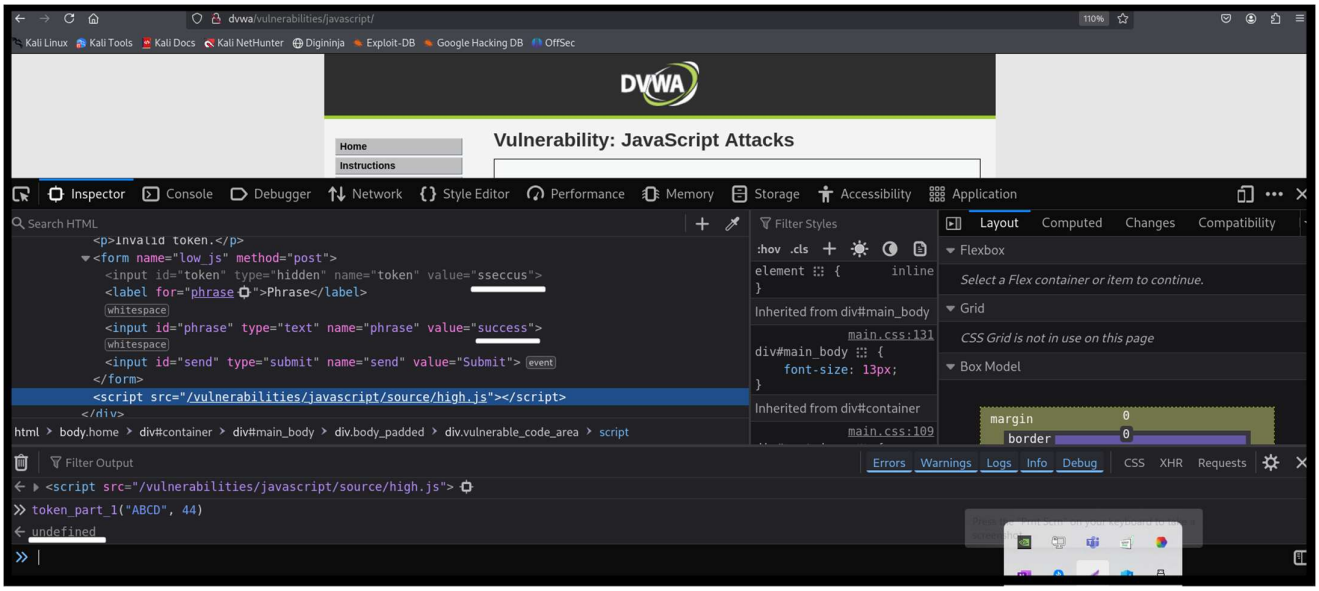
js

token_part_1("ABCD", 44);

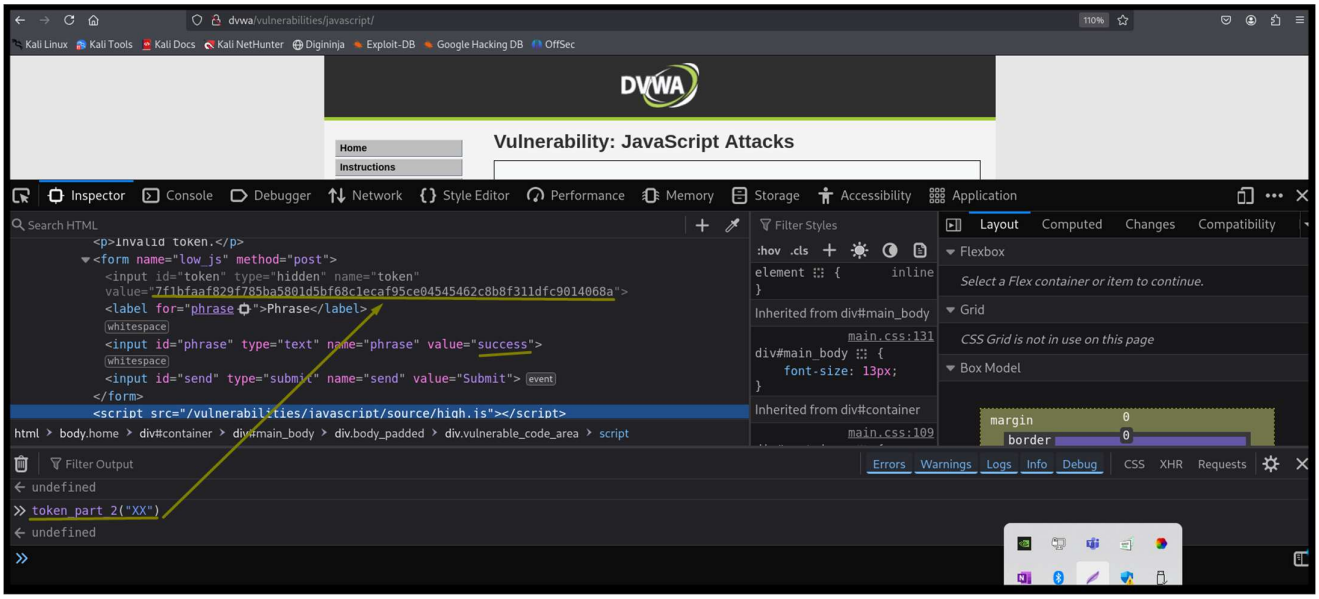
and hit enter



Step: -5 In this step we can see the token value is change success to reverse value is sseccus.



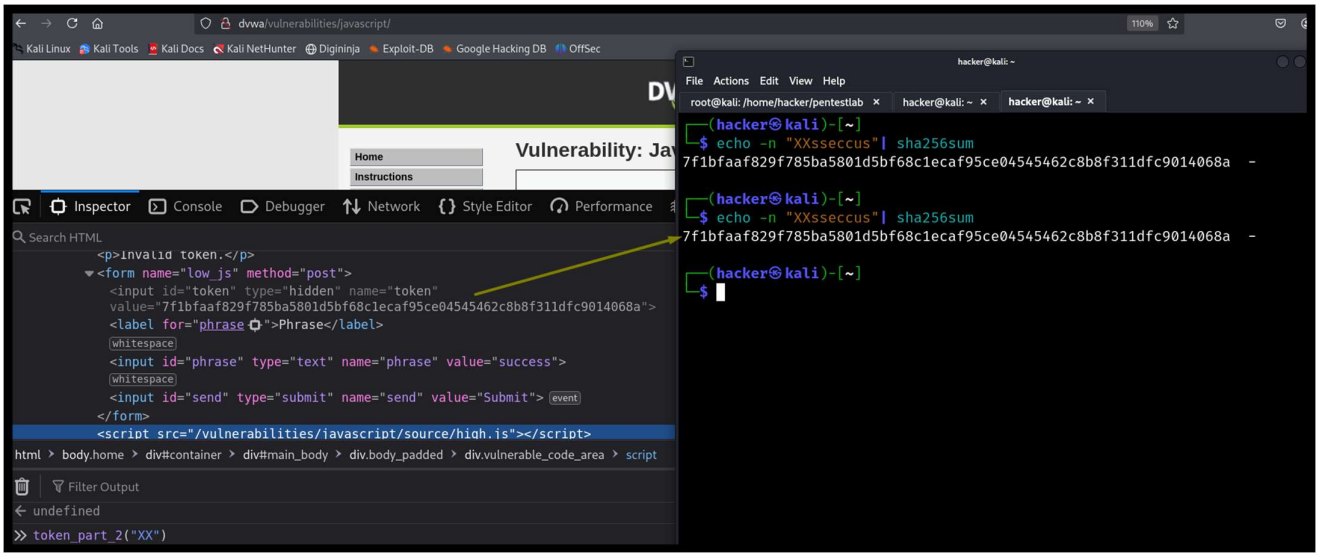
Step: -6 In this step the second next token_part_2("XX") and hit enter then the new token value is genetrated. The token value is type of sha256.



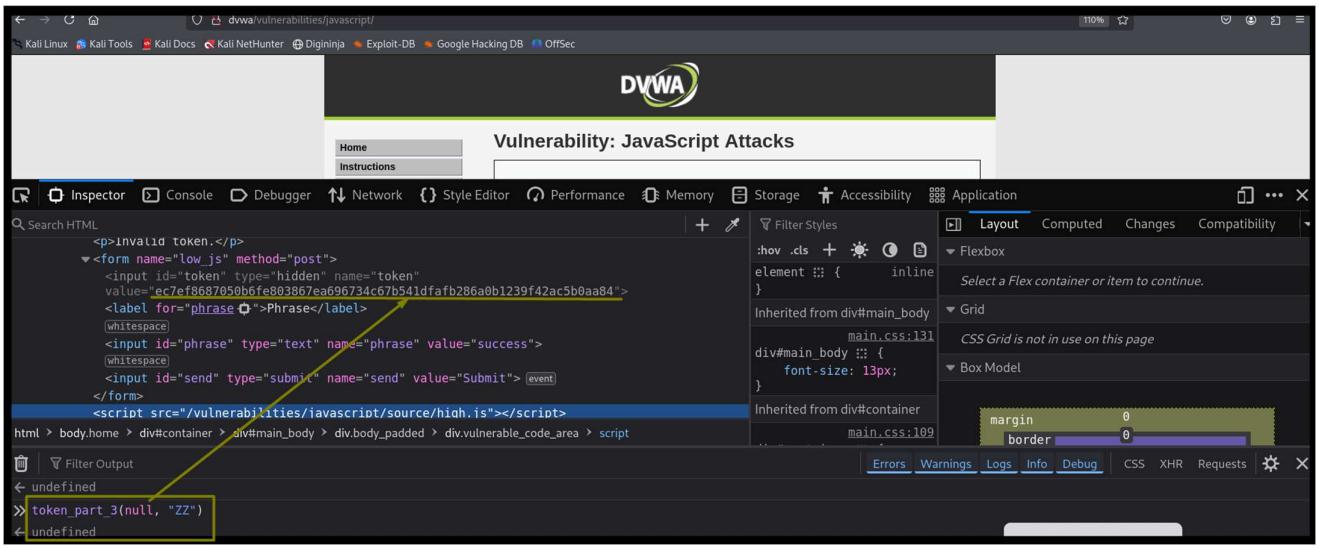
Step: -7 In this step, we can observe that both token values in the terminal are identical.

Now, use the following command in the terminal:

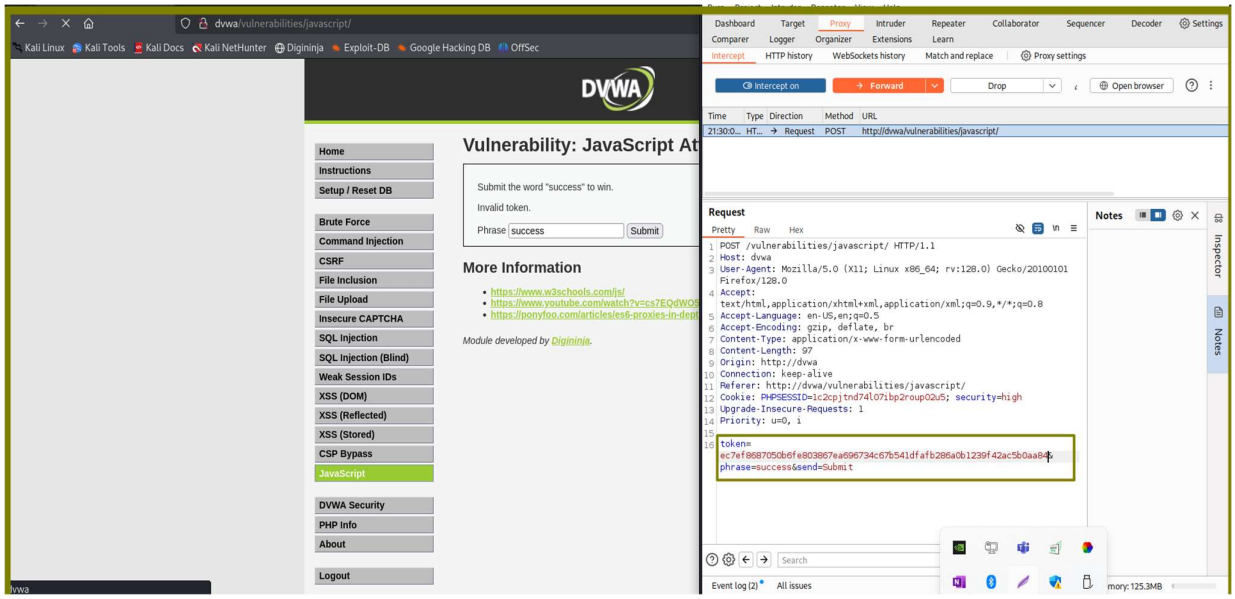
echo -n "XXsseccus" | sha256sum



Step: -8 Now we can generate the third token value. And copies this value and open burpsuite.



Step: -9 In this step, I am intercepting the request in Burp Proxy, then replacing the token value with a new token value. After that, I forward the request.



Step: -10 Finally, the word 'success' to win, and the message 'Well done' will be displayed.