

Lab1: Exploiting DOM clobbering to enable XSS

Description: This lab contains a DOM-clobbering vulnerability. The comment functionality allows "safe" HTML. To solve this lab, construct an HTML injection that clobbers a variable and uses XSS to call the alert() function.

Testing procedure and snapshot:

- Go to one of the blog posts and create a comment containing the following anchors:
`<a id=defaultAvatar name=avatar
href="cid:"onerror=alert(1)//">`
- Return to the blog post and create a second comment containing any random text. The next time the page loads, the alert() is called.

The page for a specific blog post imports the JavaScript file loadCommentsWithDomPurify.js, which contains the following code:

```
let defaultAvatar = window.defaultAvatar || {avatar: '/resources/images/avatarDefault.svg'}
```

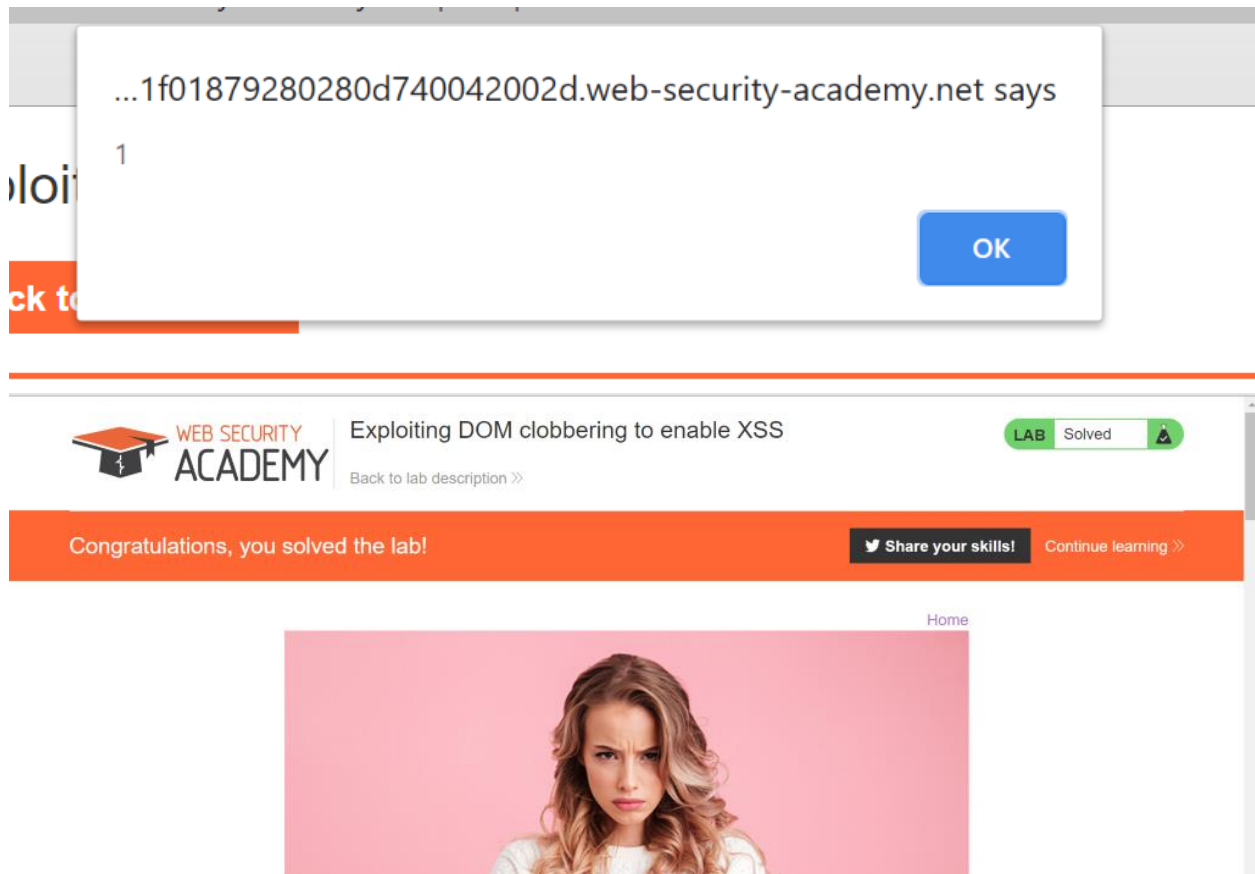
The defaultAvatar object is implemented using this dangerous pattern containing the logical OR operator in conjunction with a global variable. This makes it vulnerable to DOM clobbering.

You can clobber this object using anchor tags. Creating two anchors with the same ID causes them to be grouped in a DOM collection. The name attribute in the second anchor contains the value "avatar", which will clobber the avatar property with the contents of the href attribute.

Notice that the site uses the DOMPurify filter in an attempt to reduce DOM-based vulnerabilities. However, DOMPurify allows you to use the cid: protocol, which does not URL-encode double-quotes. This means you can inject an encoded double-quote that will be decoded at runtime. As a result, the injection described above will cause the defaultAvatar variable to be assigned the clobbered property {avatar: 'cid:"onerror=alert(1)//'} the next time the page is loaded.

When you make a second post, the browser uses the newly-clobbered global variable, which smuggles the payload in the onerror event handler and triggers the alert().

.



Lab2: Clobbering DOM attributes to bypass HTML filters

Description: This lab uses the HTMLJanitor library, which is vulnerable to DOM clobbering. To solve this lab, construct a vector that bypasses the filter and uses DOM clobbering to inject a vector that alerts document.cookie. You may need to use the exploit server in order to make your vector auto-execute in the victim's browser.

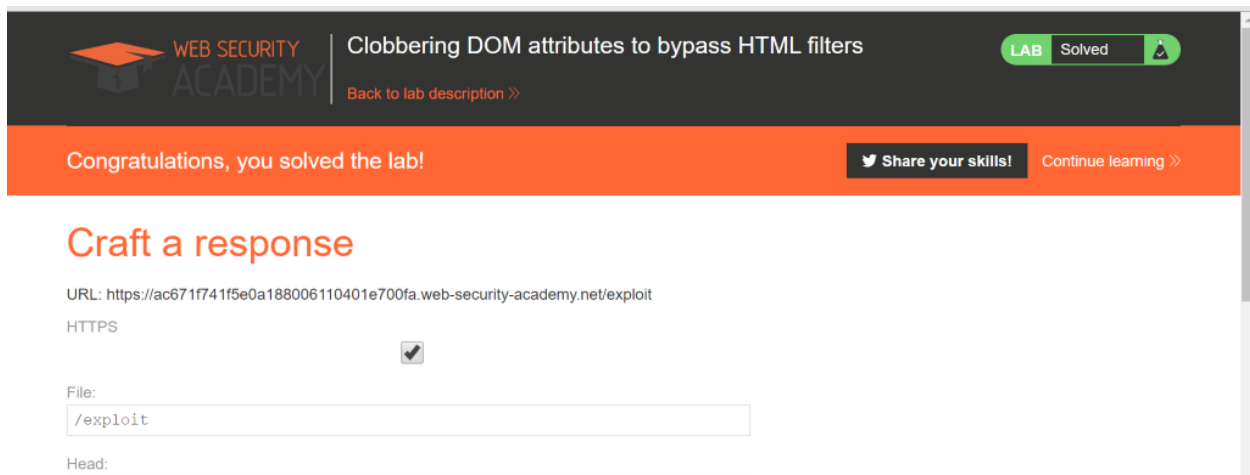
Testing procedure and snapshot:

- Go to one of the blog posts and create a comment containing the following HTML:
<form id=x tabindex=0 onfocus=alert(document.cookie)><input id=attributes>
- Go to the exploit server and add the following iframe to the body:
<iframe src=https://your-lab-id.web-security-academy.net/post?postId=3
onload="setTimeout(someArgument=>this.src=this.src+'#x',500)">. Remember to change the URL to contain your lab ID and make sure that the postId parameter matches the postId of the blog post into which you injected the HTML in the previous step.

- Store the exploit and deliver it to the victim. The next time the page loads, the alert() is called.

The library uses the attributes property to filter HTML attributes. However, it is still possible to clobber the attributes property itself, causing the length to be undefined. This allows us to inject any attributes we want into the form element. In this case, we use the onfocus attribute to smuggle an alert() function.

When the iframe is loaded, after a 500ms delay, it adds the #x fragment to the end of the page URL. The delay is necessary to make sure that the comment containing the injection is loaded before the JavaScript is executed. This causes the browser to focus on the element with the ID "x", which is the form we created inside the comment. The onfocus event handler then executes the alert() payload



The screenshot shows a web security lab interface. At the top, there's a dark header with the 'WEB SECURITY ACADEMY' logo on the left, the title 'Clobbering DOM attributes to bypass HTML filters' in the center, and a 'LAB Solved' badge on the right. Below the header is an orange banner with the text 'Congratulations, you solved the lab!' and buttons for 'Share your skills!' and 'Continue learning >'. The main content area has a heading 'Craft a response' in orange. Below this, it shows the URL 'https://ac671f741f5e0a188006110401e700fa.web-security-academy.net/exploit' and 'HTTPS'. There is a checkbox with a checkmark. Below that, there are input fields for 'File:' (containing '/exploit') and 'Head:'.

Lab3: DOM XSS using web messages and a JavaScript URL

Description: This lab demonstrates a DOM-based redirection vulnerability that is triggered by web messaging. To solve this lab, construct an HTML page on the exploit server that exploits the vulnerability and alerts document.cookie.

Testing procedure and snapshot:

- Notice that the home page contains an addEventListener call that listens for a web message. The JavaScript contains a flawed indexOf check that looks for the strings "http:" or "https:" anywhere within the web message. It also contains the sink location.href.
- Go to the exploit server and add the following iframe to the body, remembering to replace your-lab-id with your lab ID:

```
<iframe src="https://your-lab-id.web-security-academy.net/"
onload="this.contentWindow.postMessage('javascript:alert(document.cookie)//http:', '*')">
```

- Store the exploit and deliver it to the victim.

This script sends a web message containing an arbitrary JavaScript alert() payload, along with the string "http:". The second argument specifies that any targetOrigin is allowed for the web message.

When the iframe loads, the postMessage() method sends the JavaScript payload to the main page. The event listener spots the "http:" string and proceeds to send the payload to the location.href sink, where the alert() function is called in the context of the client's session.





DOM XSS using web messages and a JavaScript URL

Back to lab description >>

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning >>

Craft a response

URL: <https://acc81f261f394c0980ef264601a800f2.web-security-academy.net/exploit>

HTTPS



File:

/exploit

Head:

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

Lab4: DOM-based open redirection

Description: This lab contains a DOM-based open-redirection vulnerability. To solve this lab, exploit this vulnerability and redirect the victim to the exploit server.

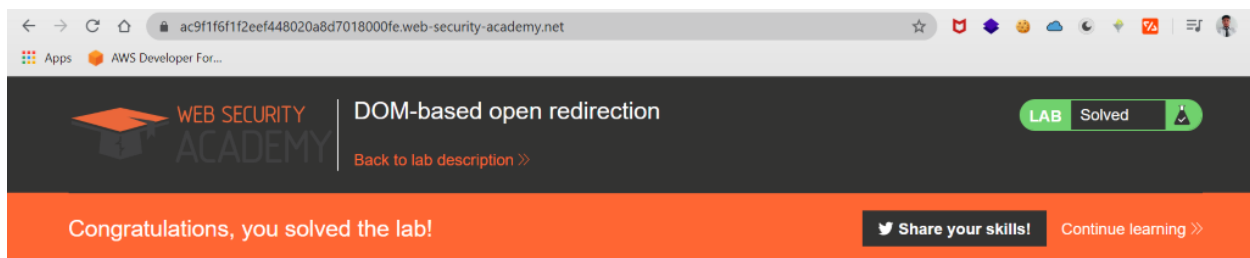
Testing procedure and snapshot:

The blog post page contains the following link, which returns to the home page of the blog:

```
<a href='#' onclick='returnURL' = /url=https?:\\\/.+)/.exec(location); if(returnUrl)location.href = returnUrl[1];else location.href = "/">Back to Blog</a>
```

The url parameter contains an open redirection vulnerability that allows you to change where the "Back to Blog" link takes the user. To solve the lab, construct and visit the following URL, remembering to change the URL to contain your lab ID and your exploit-server ID:

<https://your-lab-id.web-security-academy.net/post?postId=4&url=https://your-exploit-server-id.web-security-academy.net/>



Craft a response

URL: <https://ac9f1f6f1f2eef448020a8d7018000fe.web-security-academy.net/exploit>

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK

Lab5: DOM-based cookie manipulation

Description: This lab demonstrates DOM-based client-side cookie manipulation. To solve this lab, inject a cookie that will cause XSS on a different page. You will need to use the exploit server in order to direct the victim to the correct pages. The lab is solved when the user's document.cookie is alerted.

Testing procedure and snapshot:

- Notice that the home page uses a client-side cookie called lastViewedProduct, whose value is the URL of the last product page that the user visited.
- Go to the exploit server and add the following iframe to the body, remembering to replace your-lab-id with your lab ID:

```
<iframe src="https://your-lab-id.web-security-academy.net/product?productId=1&"><script>alert(document.cookie)</script>"  
onload="if(!window.x)this.src='https://your-lab-id.web-security-academy.net';window.x=1;">
```
- Store the exploit and deliver it to the victim.

The original source of the iframe matches the URL of one of the product pages, except there is a JavaScript payload added to the end. When the iframe loads for the first time, the browser temporarily opens the malicious URL, which is then saved as the value of the lastViewedProduct cookie. The onload event handler ensures that the victim is then immediately redirected to the home page, unaware that this manipulation ever took place. While the victim's browser has the poisoned cookie saved, loading the home page will cause the payload to execute.

...21fcfec980772ee3004700a1.web-security-academy.net says
lastViewedProduct=https://aca91f121fcfec980772ee3004700a1.web-
security-academy.net/product?
productId=1&%27%3E%3Cscript%3Ealert(document.cookie)%3C/
script%3E

OK



WEB SECURITY

ACADEMY

DOM-based cookie manipulation

[Back to lab description >>](#)

LAB

Solved



Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

Craft a response

URL: <https://ac831f981fb8ecda80ba2eb00173005e.web-security-academy.net/exploit>

HTTPS



File:

/exploit

Head:

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

Lab6: DOM XSS using web messages

Description: This lab demonstrates a simple web-message vulnerability. To solve this lab, use the exploit server to post a message to the website that exploits an XSS vulnerability and alerts document.cookie.

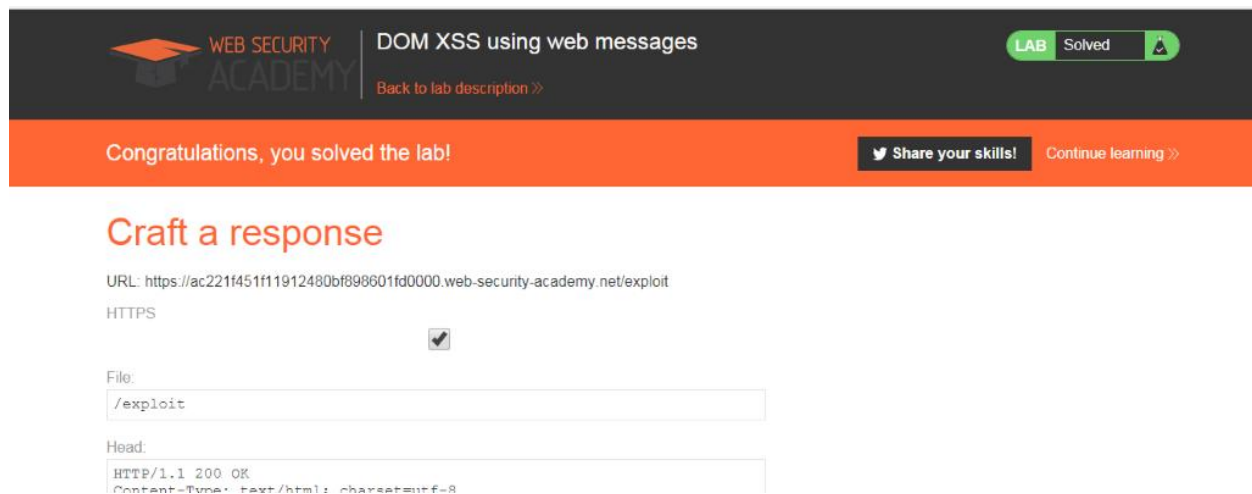
Testing procedure and snapshot:

- Notice that the home page contains an `addEventListener` call that listens for a web message.
- Go to the exploit server and add the following `iframe` to the body, remembering to change your-lab-id with your lab ID:
`<iframe src="https://your-lab-id.web-security-academy.net/"`

```
onload="this.contentWindow.postMessage('<img src=1
onerror=alert(document.cookie)>','*')">
```

- Store the exploit and deliver it to the victim.

When the iframe loads, the `postMessage()` method sends a web message to the home page. The event listener, which is intended to serve ads, takes the content of the web message and inserts it into the div with the ID `ads`. However, in this case it inserts our `img` tag, which contains an invalid `src` attribute. This throws an error, which causes the `onerror` event handler to execute our payload.



Lab7: DOM XSS using web messages and `JSON.parse`

Description: This lab uses web messaging and parses the message as `JSON`. To solve the lab, construct an HTML page on the exploit server that exploits this vulnerability and alerts `document.cookie`.

Testing procedure and snapshot:


- Notice that the home page contains an event listener that listens for a web message. This event listener expects a string that is parsed using `JSON.parse`. In the JavaScript, we can see that the event listener expects a `type` property and that the `load-channel` case of the switch statement changes the `iframe src` attribute.
- Go to the exploit server and add the following `iframe` to the body, remembering to replace `your-lab-id` with your lab ID:

```
<iframe src=https://your-lab-id.web-security-academy.net/
onload=this.contentWindow.postMessage("{\"type\":\"load-
channel\",\"url\":\"javascript:alert(document.cookie)\",\"*\"}")>
```
- Store the exploit and deliver it to the victim.


When the iframe we constructed loads, the `postMessage()` method sends a web message to the home page with the type `load-channel`. The event listener receives the message and parses it using `JSON.parse` before sending it to the switch.

The switch triggers the `load-channel` case, which assigns the `url` property of the message to the `src` attribute of the `ACMEplayer.element` iframe. However, in this case, the `url` property of the message actually contains our JavaScript payload.

As the second argument specifies that any `targetOrigin` is allowed for the web message, and the event handler does not contain any form of origin check, the payload is set as the `src` of the `ACMEplayer.element` iframe. The `alert()` function is called when the victim loads the page in their browser.

WEB SECURITY
ACADEMY

DOM XSS using web messages and `JSON.parse`

LAB Solved 

[Back to lab description >>](#)


Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

Craft a response

URL: `https://ac8e1f9d1f69755d80362f54012800c4.web-security-academy.net/exploit`

HTTPS 

File:

```
/exploit
```

Head:

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