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:

COLUMN1	COLUMN2	COLUMN3	COLUMN4	COLUMN5	COLUMN6
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

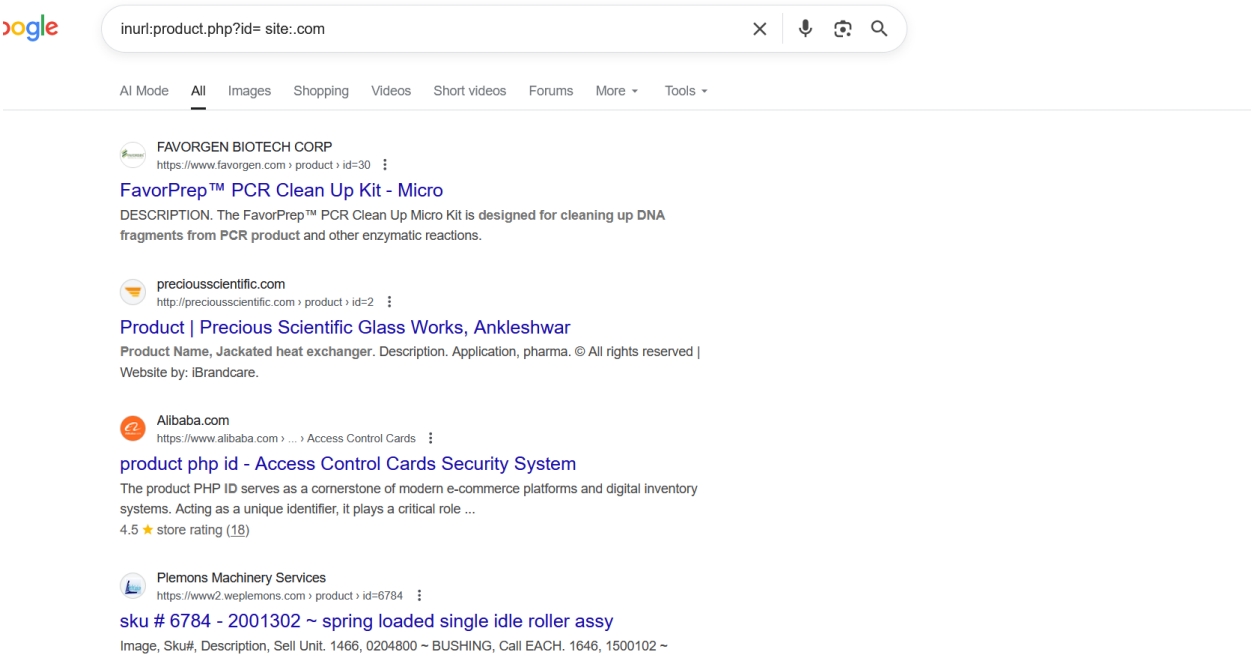
inurl:php?id=

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

site:.gov inurl:php?id=

inurl:buy.php?category=

inurl:index.php?id=



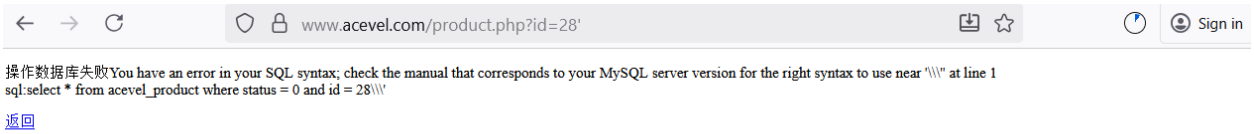
This are most sql injection dorks that i use mostly for sql injection hunting

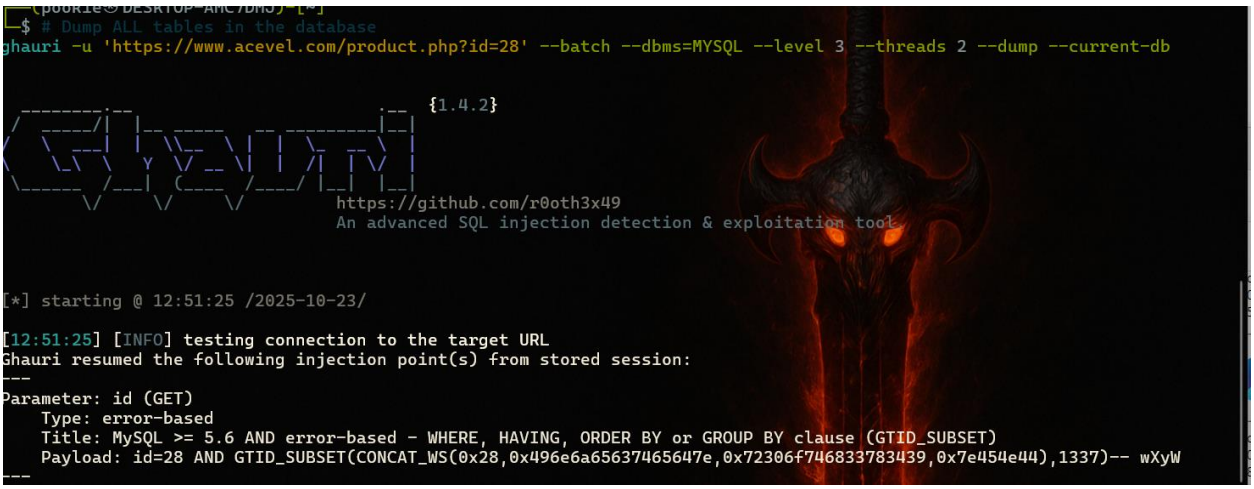
Manually test the url just look like this https://www.acevel.com/product.php?id=28 just put some sql qury in end of the url

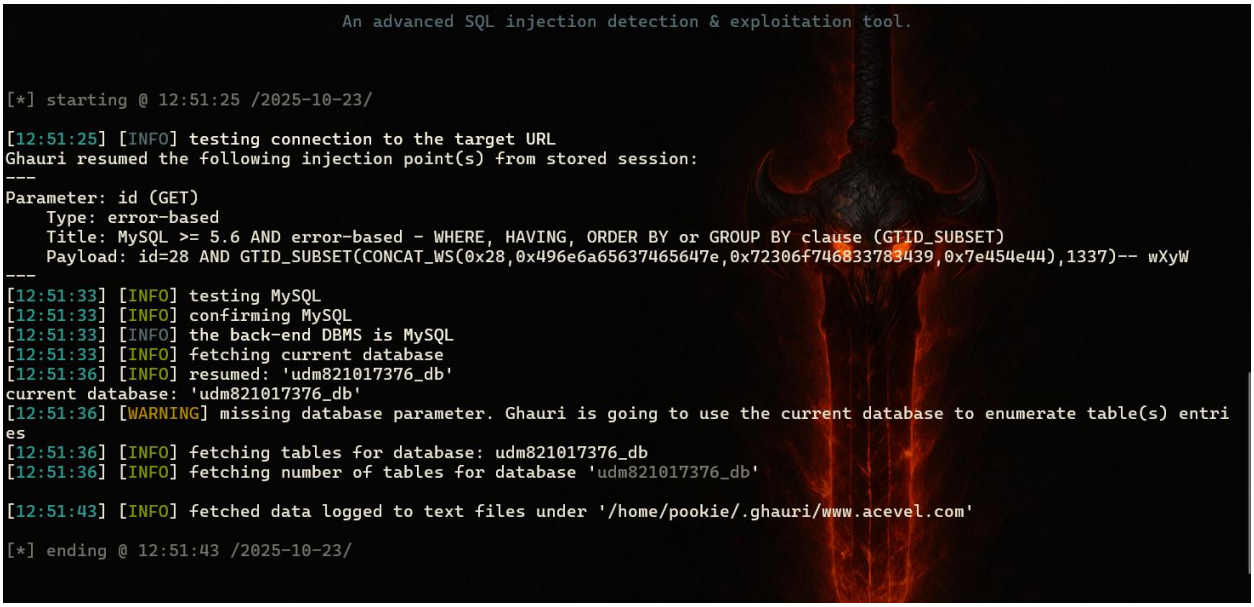
Appliy sql qury end of the url Sql qury: ‘, “, ;, --, -

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 errror below in the picture







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';
```

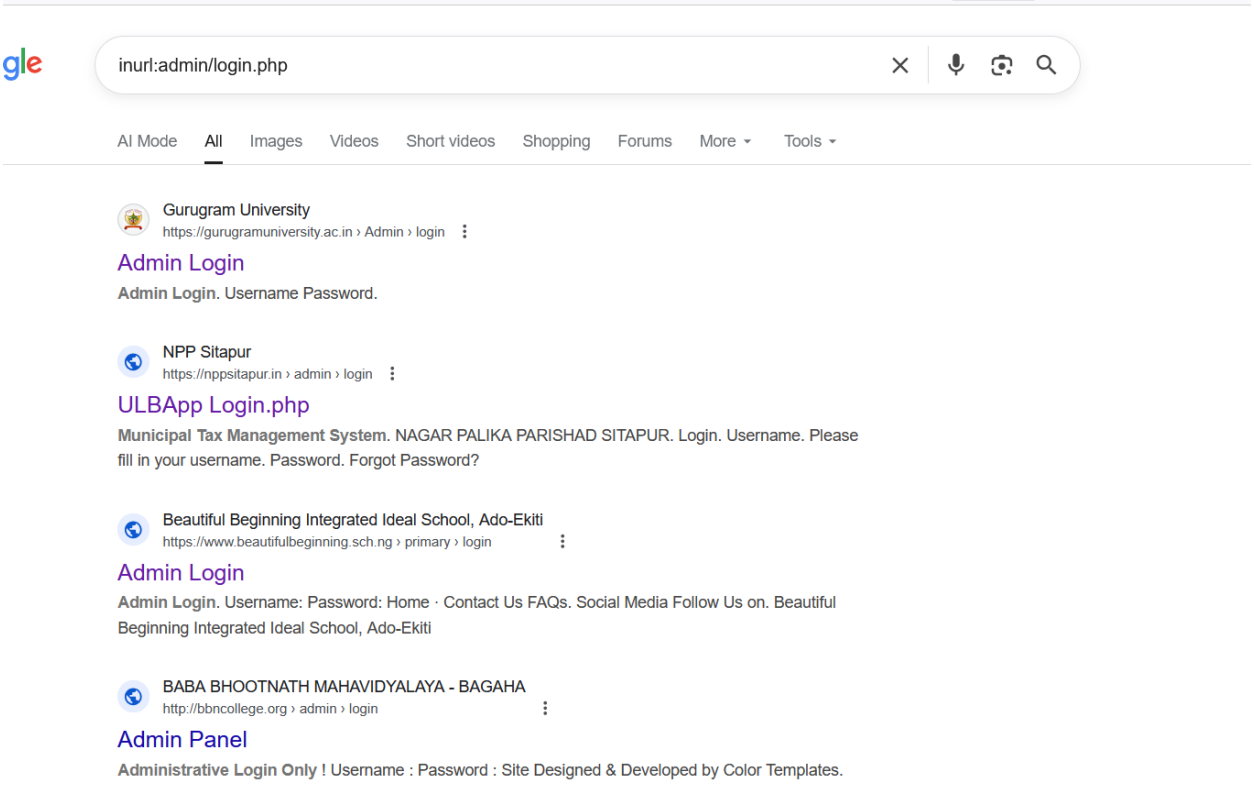
```
Username: 'or 1=1-- -
Password: any random data
```

Let’s break down what’s happening here:

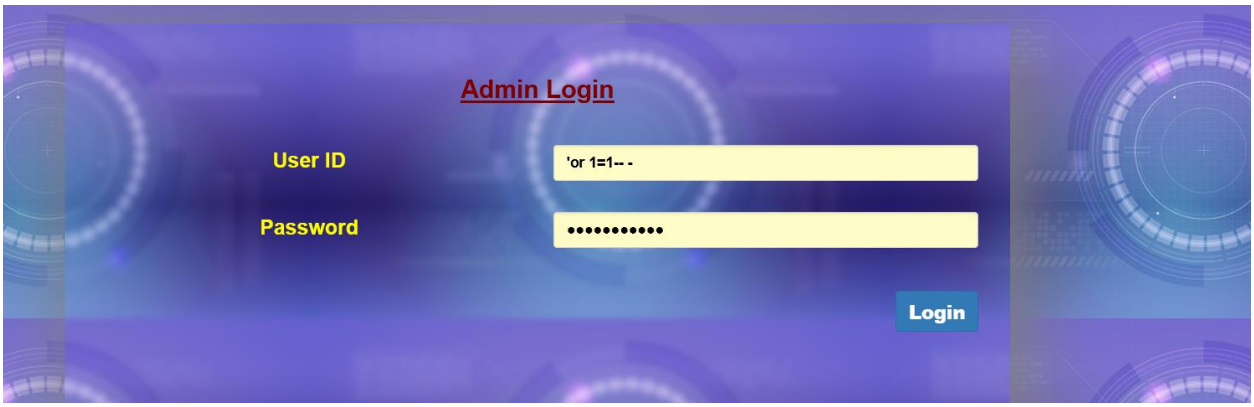
- The single quote (`'`) closes the opening quote in the SQL query, effectively closing the username field.
- `or 1=1` is always true in SQL. So, this part of the query essentially says, "Give me all the records where the username is anything or where 1 is equal to 1," which is always true. **(a=randomdata 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



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

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=gift

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 langugage) which is used to design front end Web application

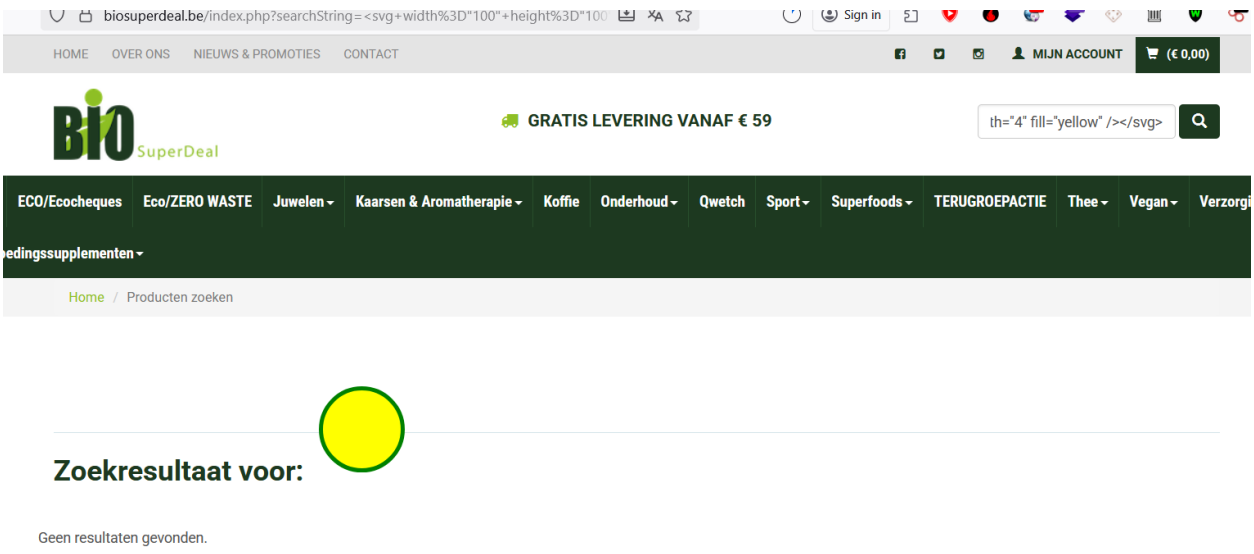
Html Injection:

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>

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



Chaining the HTML to Reflected XSS Attack:

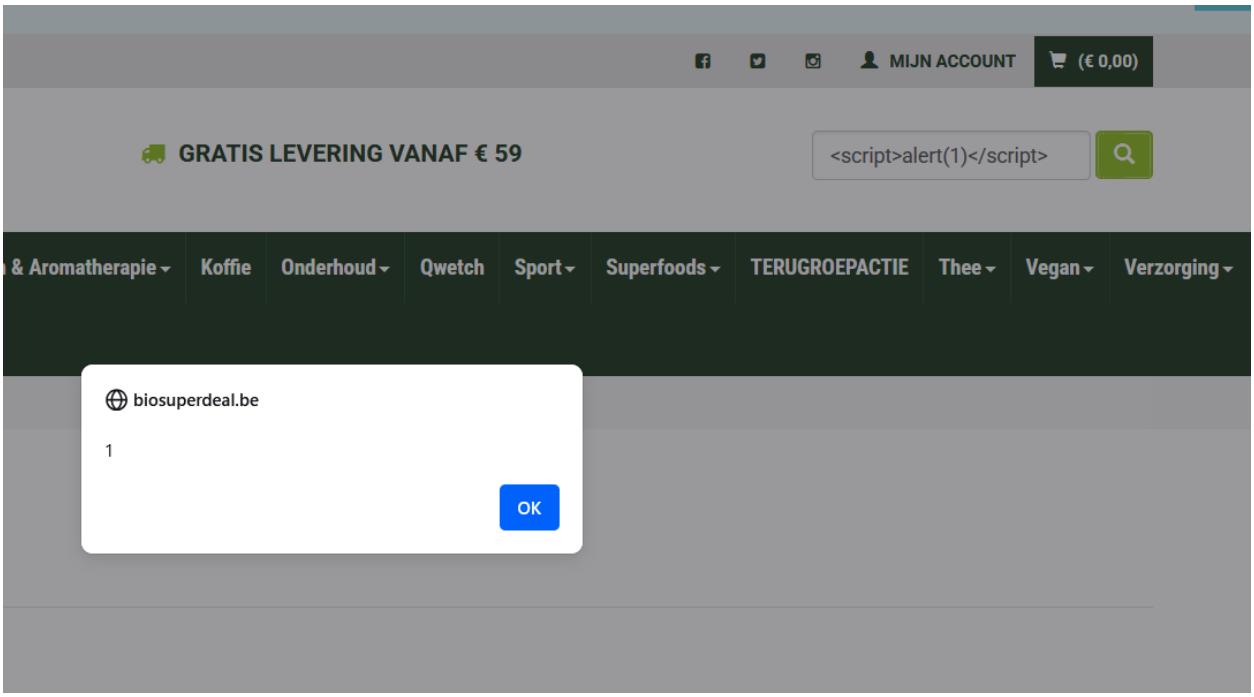
Javascript:

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



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=Carlos+Montoya&email=carlos%40normal-user.ne
```

Leave a comment

Comment:

<script>alert(1)</script>

Name:

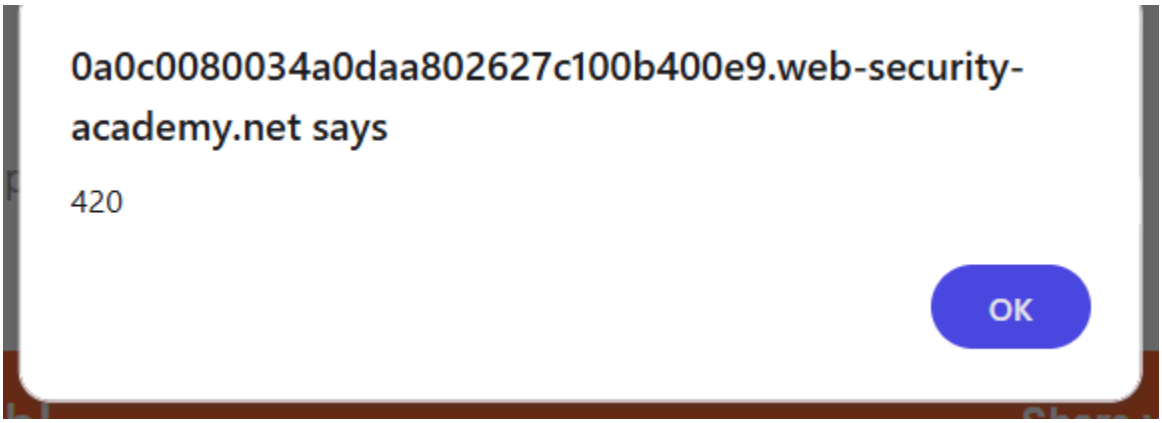
test

Email:

test@g,ail.com

Website:

test.com



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

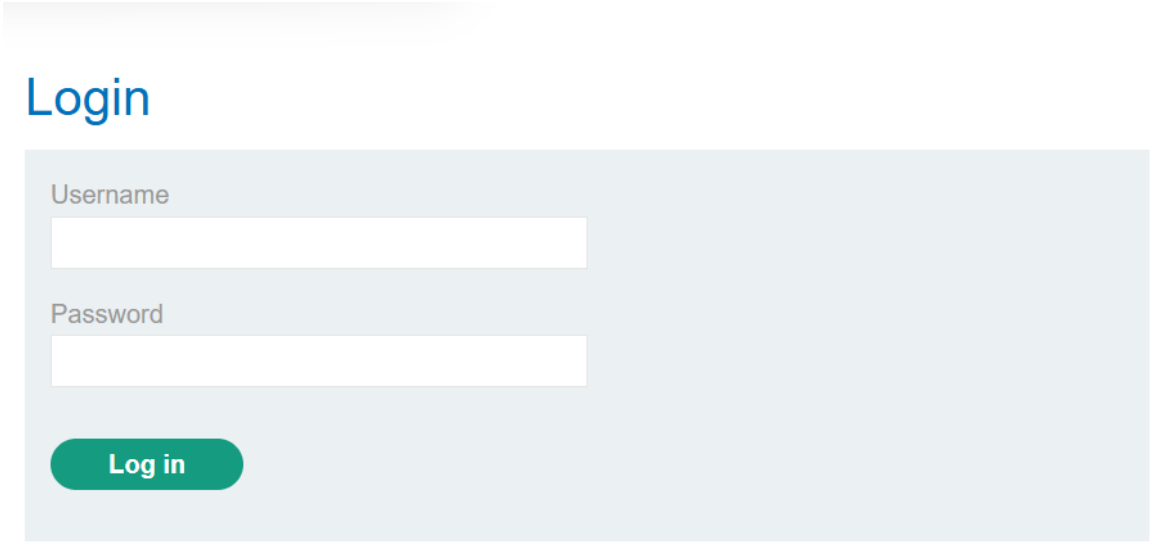
Cross-site request forgery (CSRF)

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**

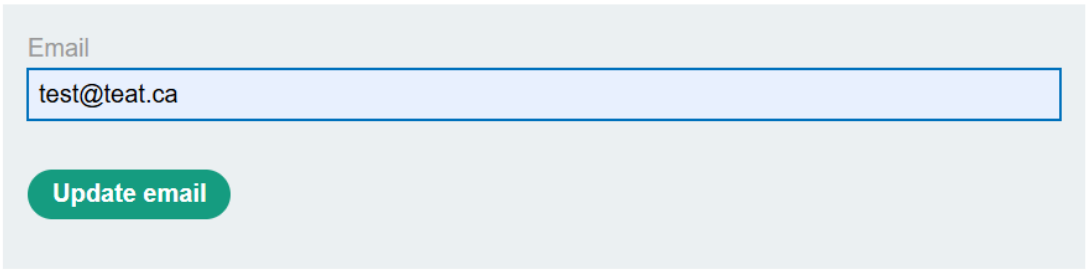
TOOL : Brupsuite professional

ATTACK METHOD: Email change functionality

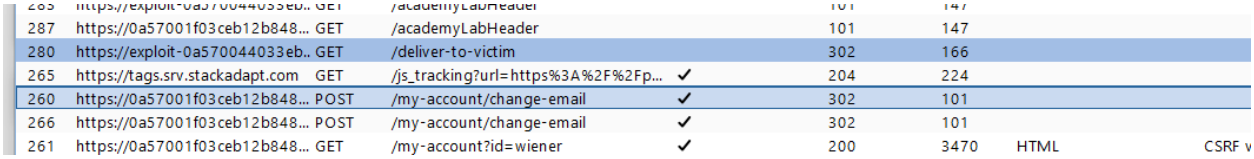
Look for login into your own account of any website



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

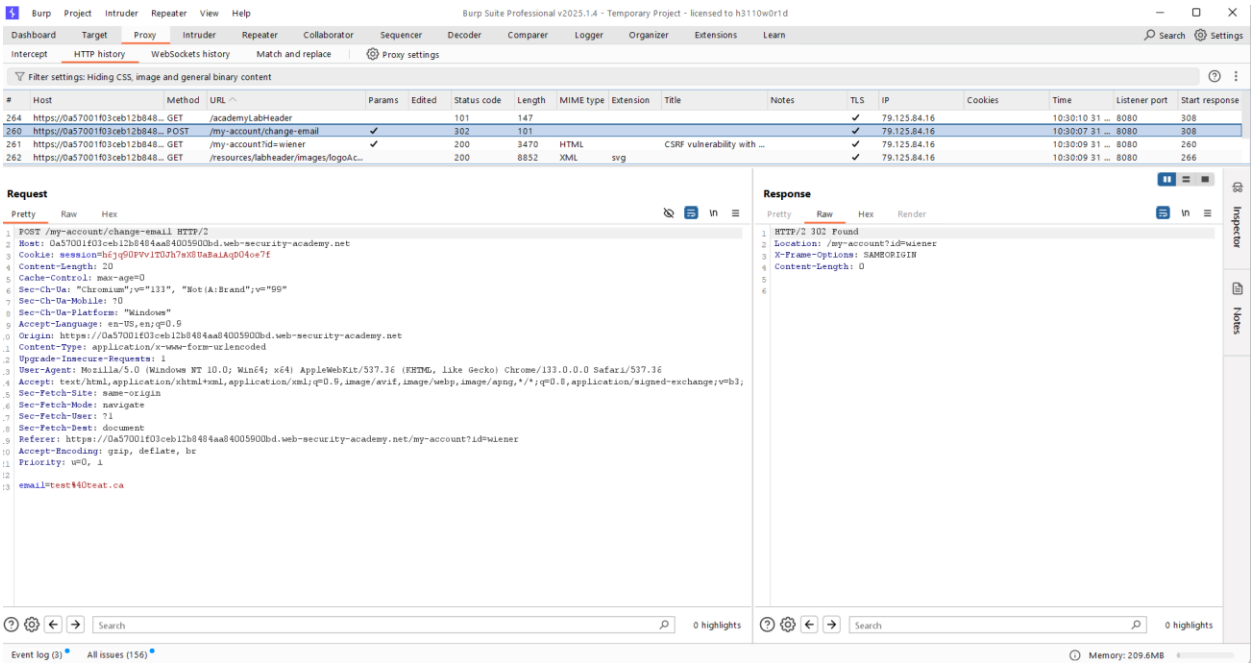


Capture that update request in brupsuite tool in

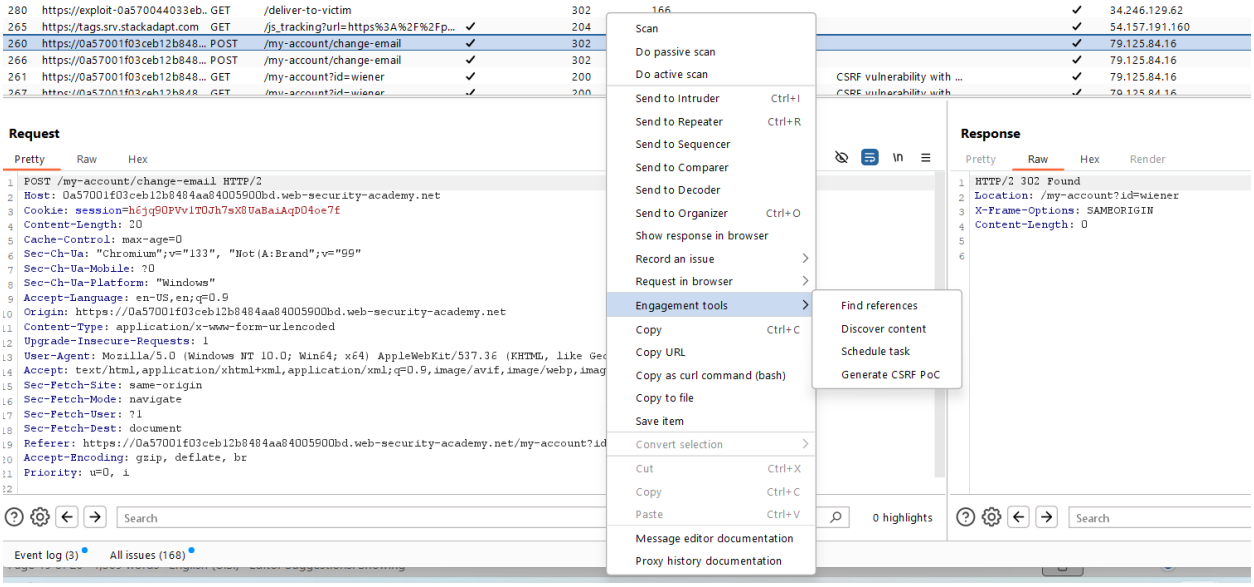


287	https://0a57001f03ceb12b848...	GET	/academyLabHeader			101	147	
280	https://exploit-0a570044033eb..	GET	/deliver-to-victim			302	166	
265	https://tags.srv.stackadapt.com	GET	/js_tracking?url=https%3A%2F%2Fp...	✓		204	224	
260	https://0a57001f03ceb12b848...	POST	/my-account/change-email	✓		302	101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

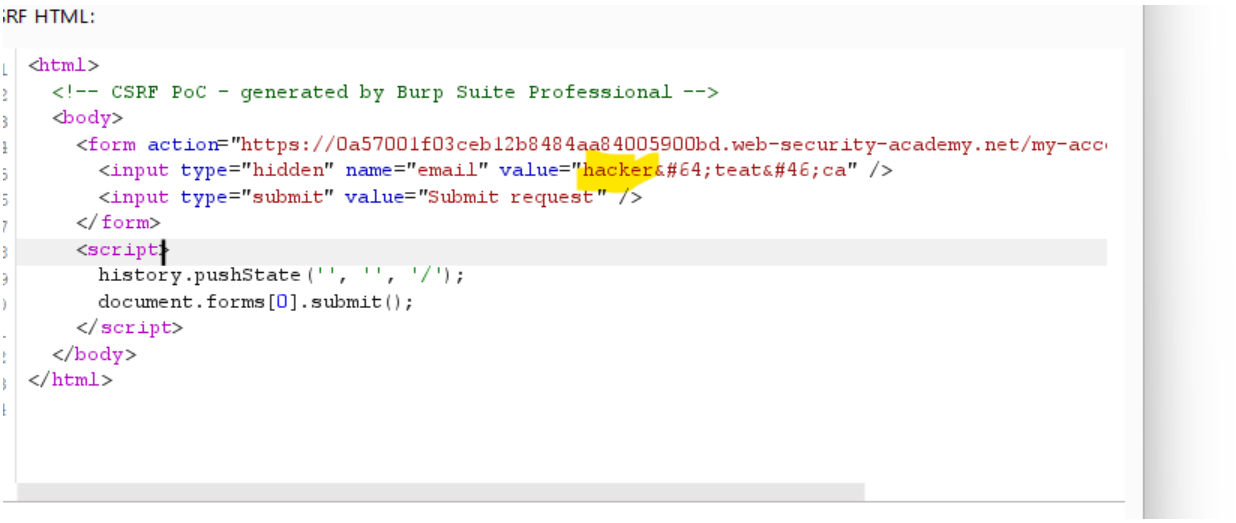
Touch on the /my-account/change-email/



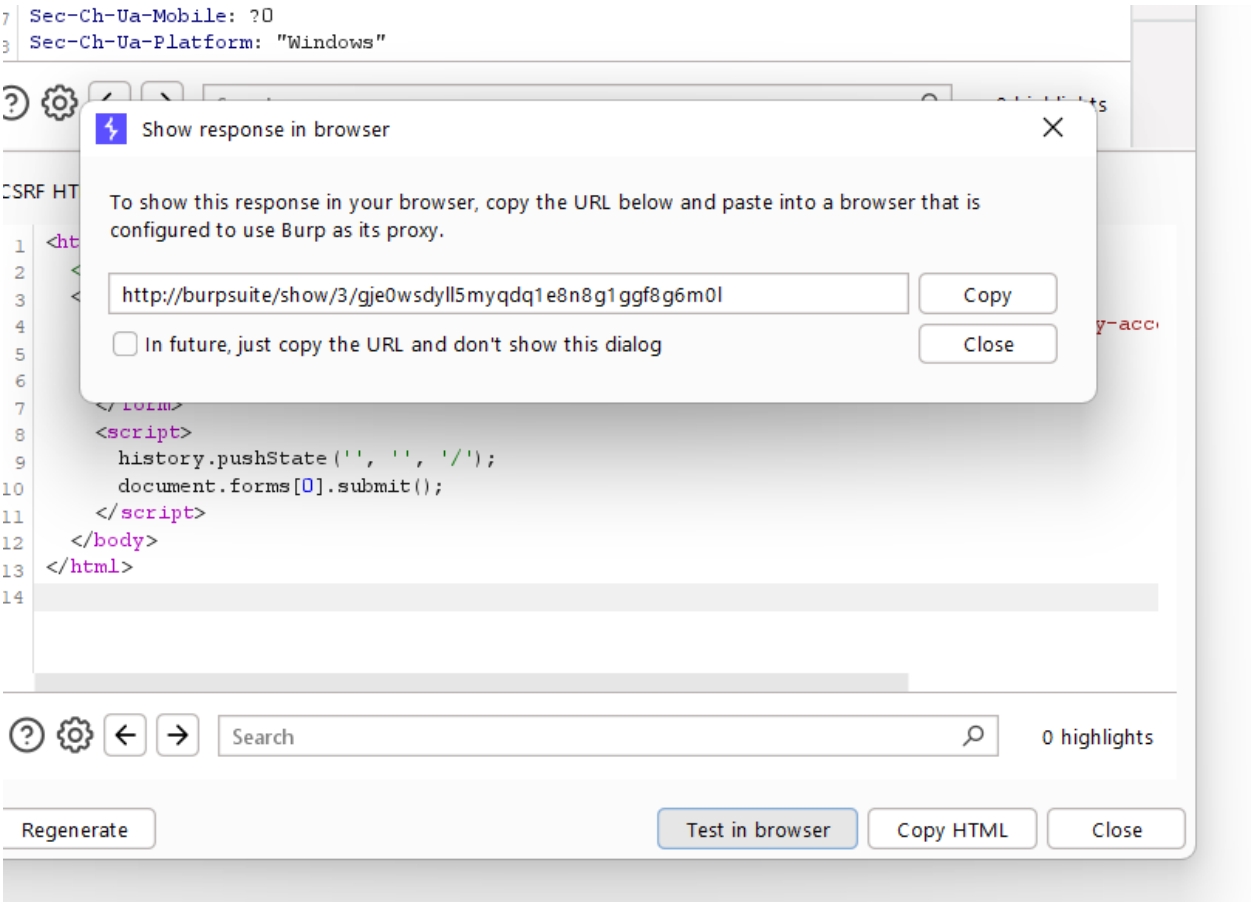
Right click on the on the request page in the below look for engage tool in Generate CSRF POC



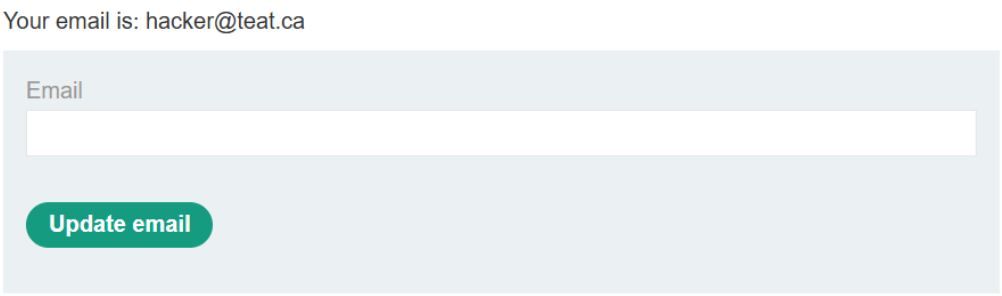
It will show the email that you updated in your account with script look for you email and try to change to hacker@gmail.com



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



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



This is the simple method to perform CSRF there is alot 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