

# Software Security HW#2

## Zehra Kolat

### SQL Injection:

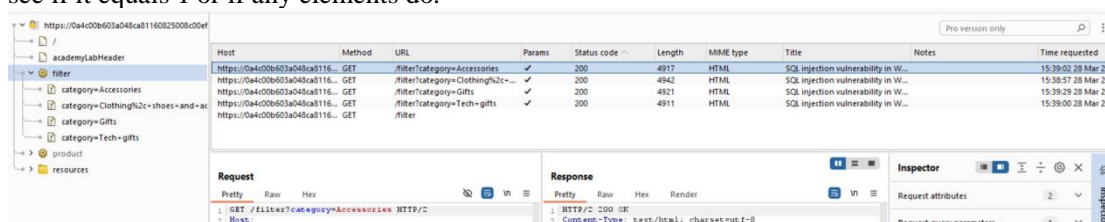
SQL injection is a critical web security vulnerability that happens when an assailant controls a websites SQL questions by infusing malevolent code into input areas, such as login shapes, look bars, or URL parameters. This helplessness emerges when client input is disgracefully sanitized and straightforwardly included in SQL statements, allowing assailants to alter database inquiries. The most causes of SQL infusion incorporate destitute input approval, need of parameterized inquiries, and lacking utilize of arranged articulations. When abused, SQL infusion can have extreme results, such as unauthorized get to to touchy information, database debasement, or indeed total control over the influenced site. Assailants can recover secret data like client accreditations, alter or erase records, and, in a few cases, execute regulatory commands on the server. To anticipate SQL infusion, engineers ought to utilize arranged explanations, parameterized questions, and input approval procedures to guarantee that client input cannot meddled with SQL execution.

### Cross-Site Scripting (XSS):

Cross-Site Scripting (XSS) could be a web security vulnerability that permits assailants to infuse pernicious scripts into web pages viewed by other clients. This happens when an online site falls flat to legitimately approve or elude client input some time recently showing it within the browser. XSS vulnerabilities are ordinarily caused by inappropriate input sanitization, need of yield encoding, and disappointment to utilize secure coding hones. There are three fundamental sorts of XSS: put away XSS, where the pernicious script is forever put away on the server; reflected XSS, where the script is included in a URL or ask and executed when a client interatomic with it; and DOM-based XSS, which happens when client-side JavaScript powerfully alters the webpage in an unreliable way. The affect of XSS assaults can be serious, counting session seizing, information robbery, phishing assaults, and the defacement of websites. To avoid XSS, engineers ought to utilize appropriate input approval, actualize yield encoding, empower Substance Security Arrangement, and dodge straightforwardly embeddings client input into the DOM without legitimate sanitization.

## Lab: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

The website in this lab has elements from the secret shop that are hidden from the average user. Our goal is to get to and expose these components. The vulnerability stems from the absence of parameterized queries, as a simple "OR 1=1--" code allows us to get around the system. Since every element in the shop is encoded with = 1, it will be shown on the screen while we check to see if it equals 1 or if any elements do.



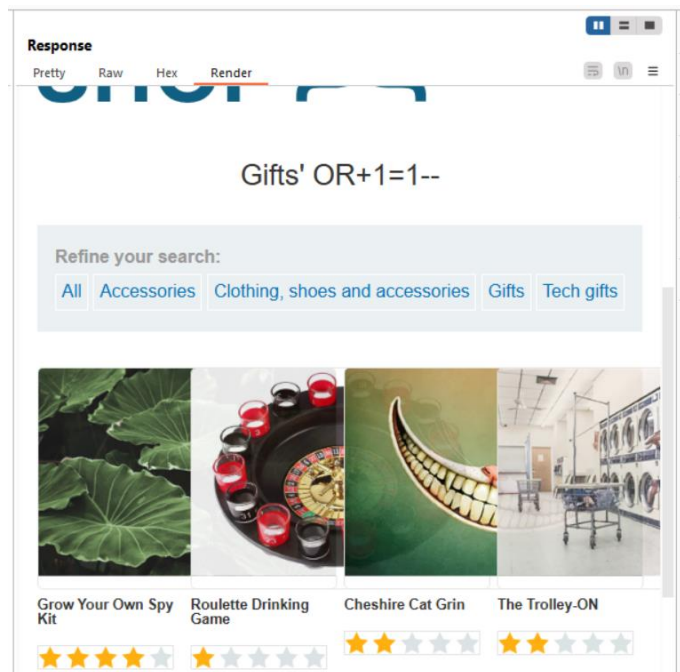
In order to access all of the backend coding and proxy the website, I utilized BurpSuite. I then checked to see whether I was receiving any internal errors. For my attacks, send it to the repeater whose URL I will alter.

## Software Security HW#2

### Zehra Kolat



I thought, after that was to try to see if I would get an internal error or not. After reading the lab instructions again and realizing that it would be released as 1, I attempted to attack by using "' OR 1=1 --" which worked and solved.



Here, we made use of the backend developer's unsanitized query check, which led to the success of our attack. We expected not to view the elements of the secret business, but we gained unlawful access. From the perspective of the business, this resulted in data leakage.

By using sanitized and appropriately parameterized queries and avoiding requiring all elements to release 1, this attack might be easily prevented.

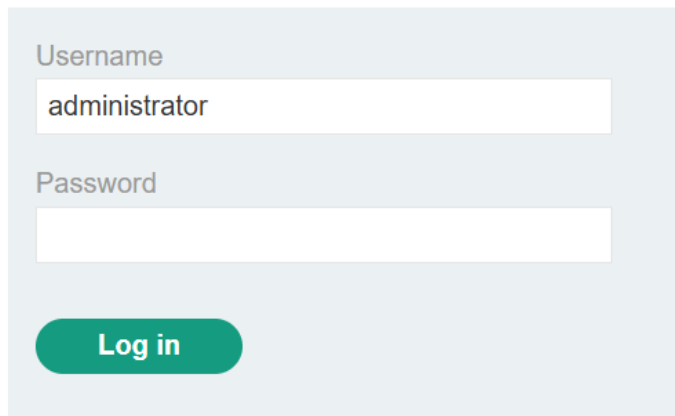
## Lab: SQL injection vulnerability allowing login bypass

The login function in this lab has a flaw that even a novice programmer could take advantage of. It doesn't operate as intended to write "administrator" in the username field. due to the fact that it has a password that is comparable to the "administrator" username.

## Software Security HW#2

### Zehra Kolat

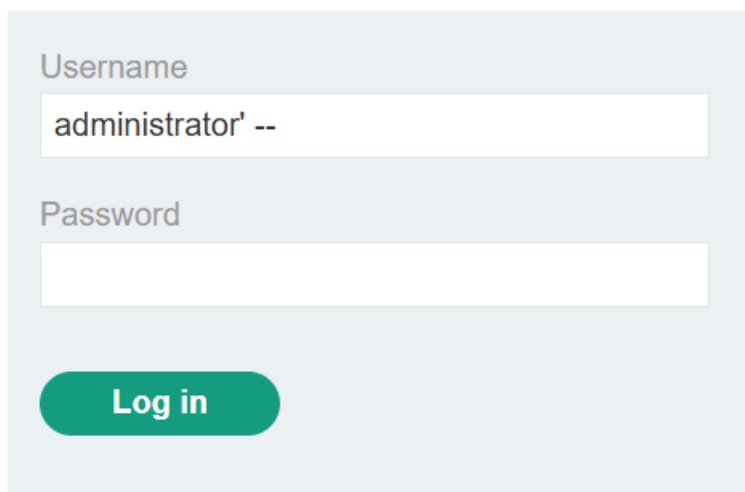
#### Login



A screenshot of a web login interface. It features a light blue background. At the top, the word "Username" is in a small, grey font. Below it is a white text input field containing the text "administrator". Underneath the username field, the word "Password" is in a small, grey font, followed by an empty white text input field. At the bottom of the form is a green rounded rectangular button with the text "Log in" in white.

I tried to get in by using administrator input and random password, it didn't work. After that i searched through ethernet finding a possible bypass, which i found as " ' -- " then tried it as "administrator' --" which eventually work.

#### Login



A screenshot of the same web login interface as above. The "Username" field now contains the text "administrator' --". The "Password" field remains empty. The green "Log in" button is still at the bottom.

## Software Security HW#2

Zehra Kolat

Congratulations, you solved the lab!

### My Account

Your username is: administrator

Email

Update email

We experienced another security breach that forced us to log in as administrators, granting us unauthorized access to the website. which could result in data theft, leaks, and other undesirable outcomes.

Properly parameterized and sanitized queries could help mitigate this exploit. Having them in the same query and being able to bypass it is the main issue with the login mechanism.

### Lab: SQL injection UNION attack, determining the number of columns returned by the query

We are retrieving data from other tables in this lab using the UNION attack, and our ultimate objective is to ascertain how many columns the query returned. which will be helpful for the upcoming attacks we plan to utilize.

//0a7600f403a8e7cf8413d6a900900065.web-security-academy.net/filter?category=Pets%27%20UNION%20SELECT%20NULL

SQL injection UNION attack, determining the number of columns returned by the query

LAB Not solved

Back to lab home

Back to lab description >>

Elements Console Sources Network

Filter

All Fetch/XHR Doc CSS JS Font Img Media

100 ms 200 ms 300 ms 400 ms

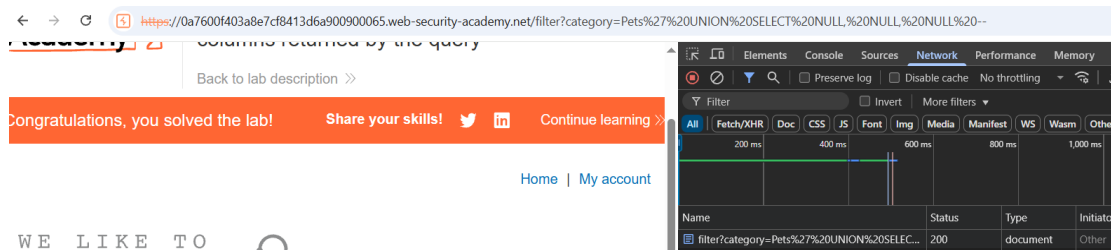
Name	Status
academyLabHeader	101
filter?category=Pets%27%20UNION%20SELEC...	500

I used google's devtools for this application to see if there is a error or not.

I started with UNION select and continue until i found the correct amount of columns.

# Software Security HW#2

Zehra Kolat



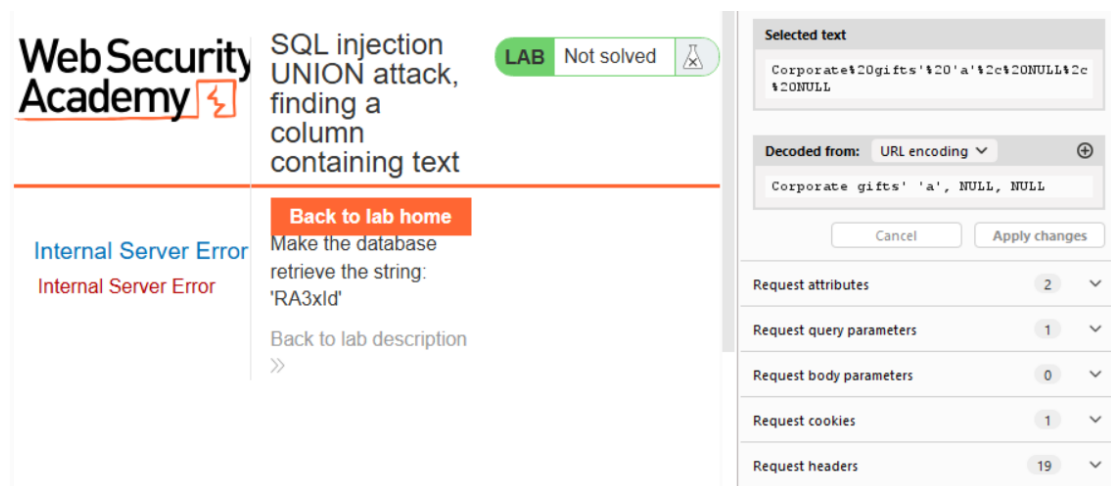
Which worked on the 3<sup>rd</sup> and after that the devtools was status also 200. Lab was solved.

This website was susceptible to SQL injection due to improperly parameterized queries and unclean backend coding.

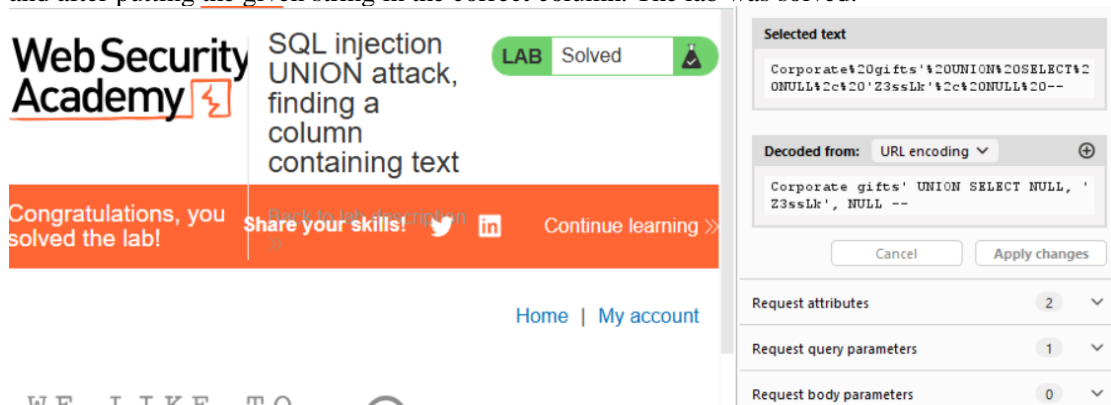
Much more focused and sterile questioning can help to lessen this. use several parameters for various tasks rather than combining them into a single parameter.

## Lab: SQL injection UNION attack, finding a column containing text

In order to exploit a website using the UNION attack once more, we must first determine how many columns it contains. Then, we will insert strings one at a time to identify the columns that contain text.



Used Burp Suite for this and started to search for columns containing string, which was the second and after putting the given string in the correct column. The lab was solved.



This website was susceptible to SQL injection due to improperly parameterized queries and unclean backend coding.

## Software Security HW#2

### Zehra Kolat

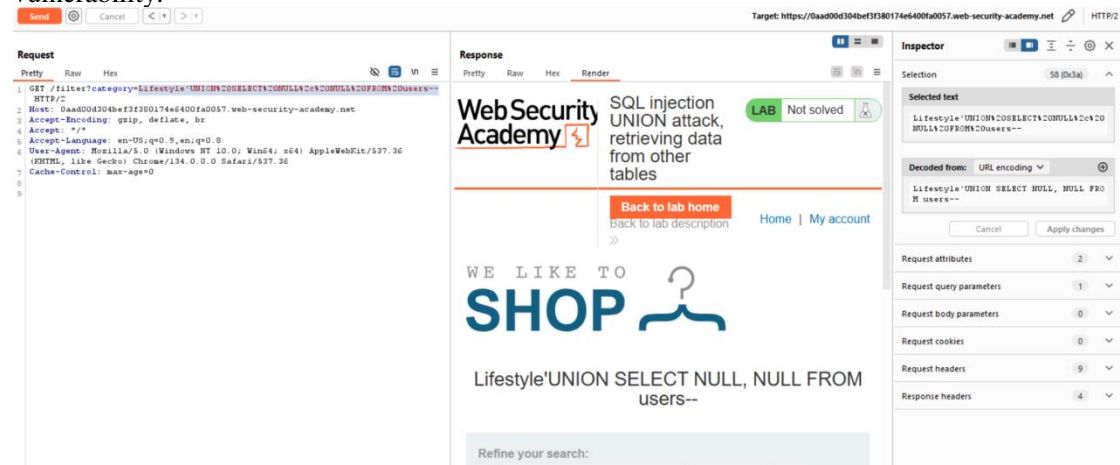
Much more focused and sterile questioning can help to lessen this. use several parameters for various tasks rather than combining them into a single parameter.

This website was susceptible to SQL injection due to improperly parameterized queries and unclean backend coding.

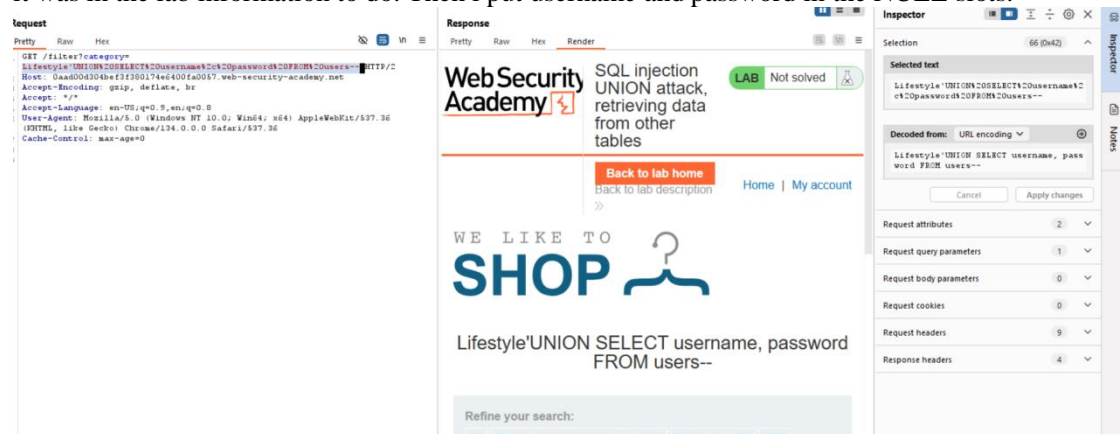
Much more focused and sterile questioning can help to lessen this. use several parameters for various tasks rather than combining them into a single parameter.

## Lab: SQL injection UNION attack, retrieving data from other tables

To access table information in this experiment, we once more employ the UNION attack. We must log in as the administrator account and output usernames and passwords in the users table. We must first ascertain how many columns the susceptible query uses and what kind of data each column contains. Once more, poorly parameterized and unsanitized queries are the source of the vulnerability.



Again, used Burp Suite, and started with the same as the before code. I started with UNION select and continue to find column number, two was the correct amount and added FROM users because it was in the lab information to do. Then i put username and password in the NULL slots.



Then i got the usernames and passwords of administrator, then put it in to the login function to work it out. It worked as intended and lab was solved.

# Software Security HW#2

## Zehra Kolat



SQL injection UNION attack, retrieving data from other tables

LAB Solved

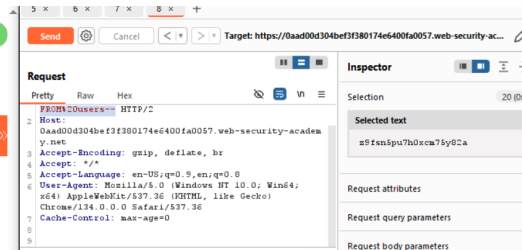
[Back to lab description >>](#)

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#) [Continue learning >>](#)

[Home](#) | [My account](#) | [Log out](#)

[My Account](#)



Inadequately parameterized queries and dirty backend coding made this website vulnerable to SQL injection.

This can be mitigated by much more sterile and focused inquiry. Instead of combining several parameters into one, use multiple parameters for different tasks.

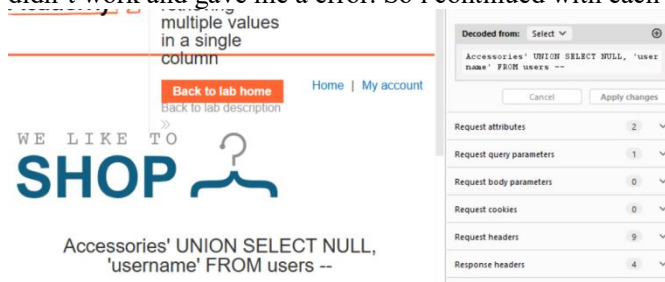
## Lab: SQL injection UNION attack, retrieving multiple values in a single column

There is a flaw in the product category filter in this lab. We will use a UNION attack to take advantage of the query's vulnerability once more. In a departure from the norm, we will utilize a separate table to retrieve the username and password, which are located in a single column.

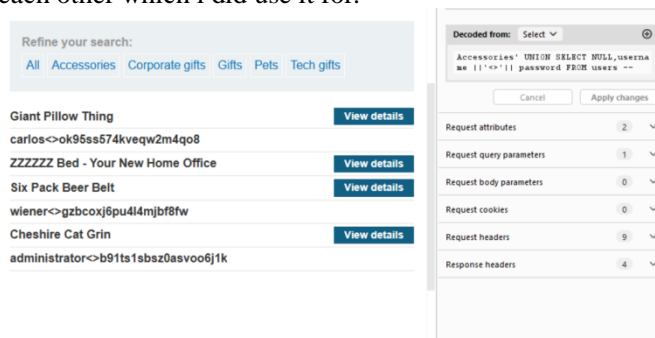
Used Burp Suite again to solve this lab. I started with “ “ to check the vulnerability. Then go on with the expected “UNION select NULL,NULL FROM users--”



Which was working. So, i found the second NULL was the containing the string after the first one didn't work and gave me a error. So i continued with each username and password.



It was working but i couldnt get both from them so, i remembered that in C you can add them to each other which i did use it for.



After that i entered the administrator and it password to login and it worked. Lab solved.

## Software Security HW#2

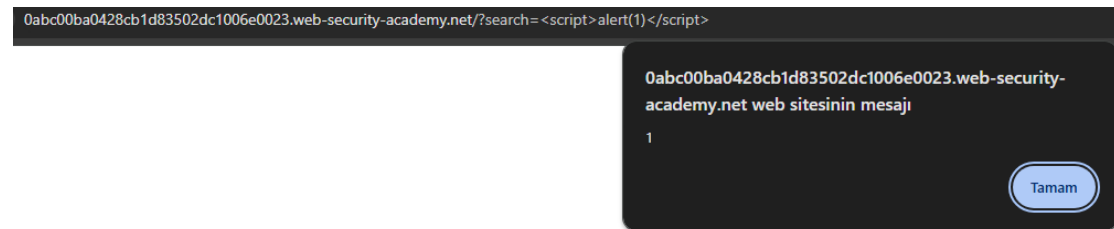
### Zehra Kolat

Inadequately parameterized queries and dirty backend coding made this website vulnerable to SQL injection.

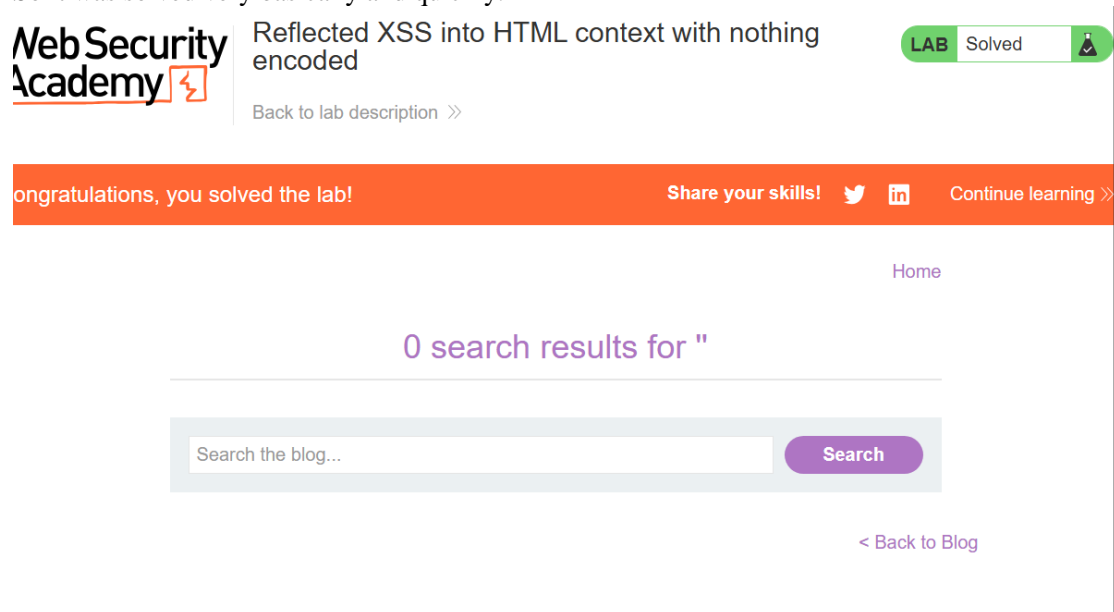
This can be mitigated by much more sterile and focused inquiry. Instead of combining several parameters into one, use multiple parameters for different tasks.

### Lab: Reflected XSS into HTML context with nothing encoded

Calling an alert function by exploiting a vulnerability in the search functionality. The attack based on the most recent http request will be reflected in our attack.



I verified that the search bar could insert script and special characters, and it could. made sure that because it passes through the and into the website, it is susceptible to manipulation by the search bar. aren't displayed on the internet either, which is proof that it actually passed. So it was solved very basically and quickly.



Here, we took advantage of a weakness in the backend Java script. The search bar and the parameters it uses to apply the search have a flaw. Special characters is the problem of this website. Mitigation would be based on special characters.

### Lab: Stored XSS into HTML context with nothing encoded

We will launch an alert function by exploiting a vulnerability in the comment functionality. The assault based on the most recent http request will be reflected in our attack. Looks like before, but this is stored in the database instead of one time thing.



## Software Security HW#2

### Zehra Kolat

#### Leave a comment

Comment:

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

Name:

zehra

Email:

zehra@kolat

Website:

https://zehrakolat.com

Post Comment

[Back to Blog](#)

This time we are using comment box to store our alert attack, as you can see on the below. This code will bypass the parameters which are wrongfully placed. And attack will be successful.

Congratulations, you solved the lab!

Thank you for your comment!

Your comment has been submitted.

There is a problem with the special characters in the comment box, it shouldn't pass them by into the backend, but it does because of the poorly written parameters. Mitigation would be correcting these parameters.

### Lab: DOM XSS in document.write sink using source location.search

We are using DOM XSS to take advantage of a flaw in the search query tracking mechanism. We modify the code and invoke the document.write function using location.search, which causes the program to write data to the page.

## Software Security HW#2

### Zehra Kolat

Used Devtools again to find document.write and possible vulnerabilities.

```
function trackSearch(query) {
    document.write('
```

Which i found like this. I realized everything that i write in the search bar, goes uncensoredly and directly to the query and writed with the document.write function to the DOM. I searched and find onload function to bypass and have access to the html coding. I used it.

0aa0000c0374993c8028d0ee00c4003a.web-security-academy.net/?search=\*><svg+onload%3Dalert%281%29>

Security emy

DOM XSS in document.write sink us location.search

Back to lab description >>

0aa0000c0374993c8028d0ee00c4003a.web-security-academy.net web sitesinin mesaji

1

Tamam

Home

0 search results for '"><svg onload=alert(1)>'

Which solved the lab resultingly. It was quite the experience.

Here, we took advantage of the imgsrc method, which generates a new query based on our search and scans the content. We can alter the program to alert ourselves by writing to the database in accordance with the write function.

It can be lessened by using search bars, which are far safer than writing straight on the database before verifying the search. There ought to be far more accurate parameterized methods for verifying special characters.

Thanks, Zehra Kolat.