

存储型 XSS 攻击实战

1. Low 级别存储型 XSS 攻击实战

步骤 1：安全级别设置为 Low，点击 XSS (Stored)，进入存储型 XSS 攻击页面。发现该页面是个留言板，随意输入留言内容，可以直接显示在当前页面，如图 1.1。



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: ccb
Message: hello

图 1.1

步骤 2：尝试在 Message 框提交弹窗脚本输出当前 cookie，
<script>alert(document.cookie)</script>，可以直接弹窗，如图 1.2。说明当前级别没有对攻击脚本做任何过滤和转义。

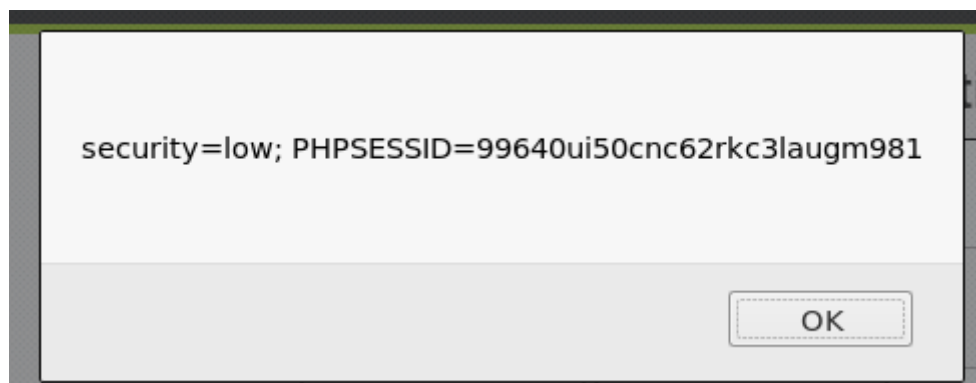


图 1.2

2. Medium 级别存储型 XSS 攻击实战

步骤 1：安全级别设置为 Medium，进入存储型 XSS 攻击页面，直接在 Message 框提交输出 cookie 脚本，发现脚本内容被显示，说明脚本被转义，如图 2.1。



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: ccb
Message: alert(document.cookie)

图 2.1

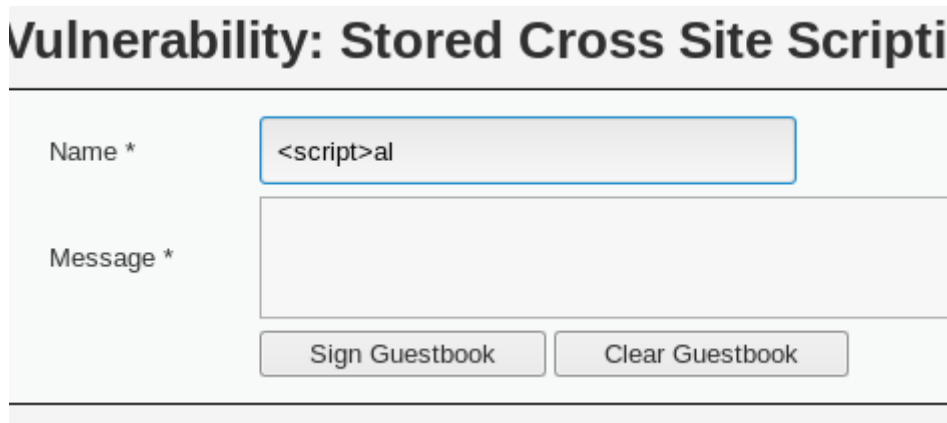
步骤 2：查看页面源码，发现对 Message 框提交的内容使用了 `htmlspecialchars` 函数，Message 框的 XSS 基本已不可能；但是我们发现对 Name 框提交的内容只是简单的使用 `str_replace` 函数进行了简单替换，与 Medium 反射型 XSS 一样，如图 2.2。我们只需要更换大写字母，或者 `<script>` 中再嵌套一层 `<script>` 即可绕过防御。



```
if(isset($_POST['btnSign'])) {  
    // Get input  
    $message = trim($_POST['mtxMessage']);  
    $name = trim($_POST['txtName']);  
  
    // Sanitize message input  
    $message = strip_tags addslashes($message);  
    $message = ((isset($GLOBALS['__mysqli_ston']) && is_object($GLOBALS['__mysqli_ston'])) ? mysqli_real  
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : '  
    $message = htmlspecialchars($message);  
  
    // Sanitize name input  
    $name = str_replace('<script>', '', $name);  
    $name = ((isset($GLOBALS['__mysqli_ston']) && is_object($GLOBALS['__mysqli_ston'])) ? mysqli_real  
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : '  
  
    // Update database  
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";  
    $result = mysqli_query($GLOBALS['__mysqli_ston'], $query) or die('<pre>' . ((is_object($GLOBALS['__mysqli_ston']) && is_object($GLOBALS['__mysqli_ston'])) ? mysqli_real  
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) . "</pre>";  
}
```

图 2.2

步骤 3: 在 Message 框随意输入内容, Name 框输入脚本
<script>alert(document.cookie)</script>, 发现脚本无法完整输入, 页面对 Name 框的字符长度进行了前端限制, 如图 2.3。



Vulnerability: Stored Cross Site Scripting

Name *

Message *

图 2.3

步骤 4: 由于只是在浏览器前端进行的字符长度限制, 我们只需要在 Burpsuite 中修改数据包就可以轻松绕过限制。配置好 Burpsuite 和浏览器的代理, 抓包后, 修改 txtName 变量的值为脚本
<script>alert(document.cookie)</script> (也可以把 script 转换为大写), 如图 2.4。放行数据包后, 可以成功提交, 并弹窗输出 cookie, 如图 2.5。

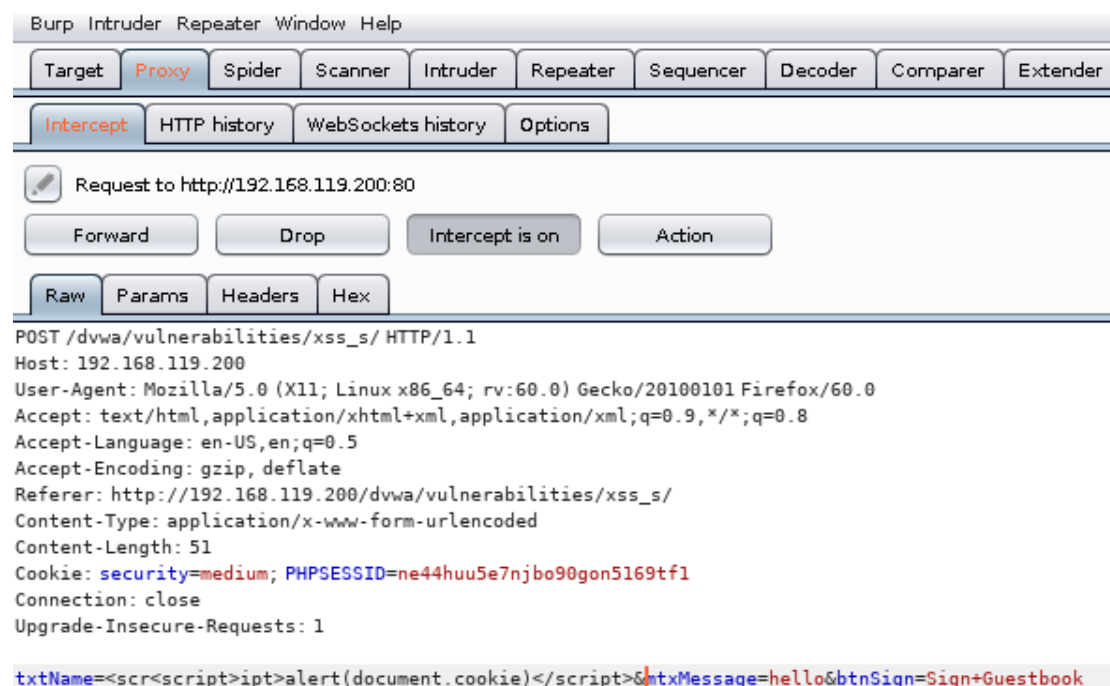


图 2.4

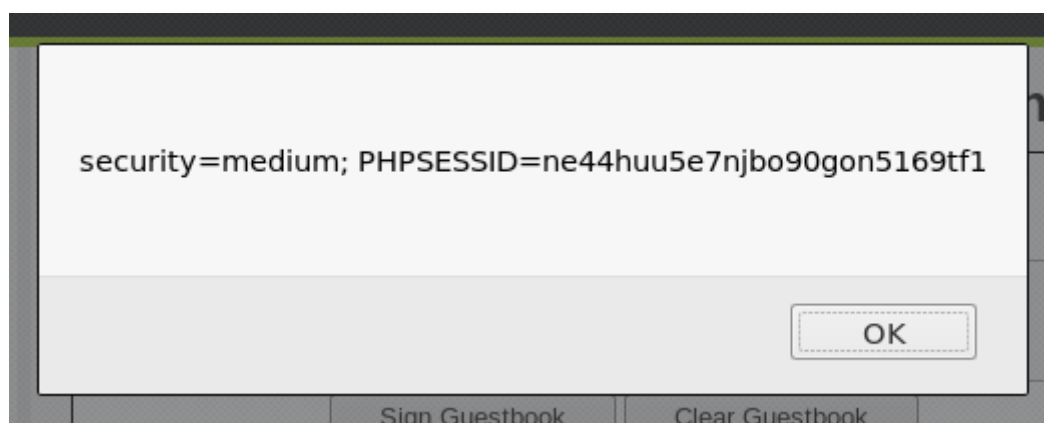


图 2.5

3. High 级别存储型 XSS 攻击实战

步骤 1：设置安全级别为 High，进入存储型 XSS 攻击页面，查看页面源码，发现 Message 字段仍然使用了 htmlspecialchars 函数；而 Name 字段使用了与 High 级别反射型 XSS 攻击一样的防护方法，使用 preg_replace 函数调用正则表达式对 script 做逐字匹配，并使用 /i 来不区分大小写，如图 3.1。

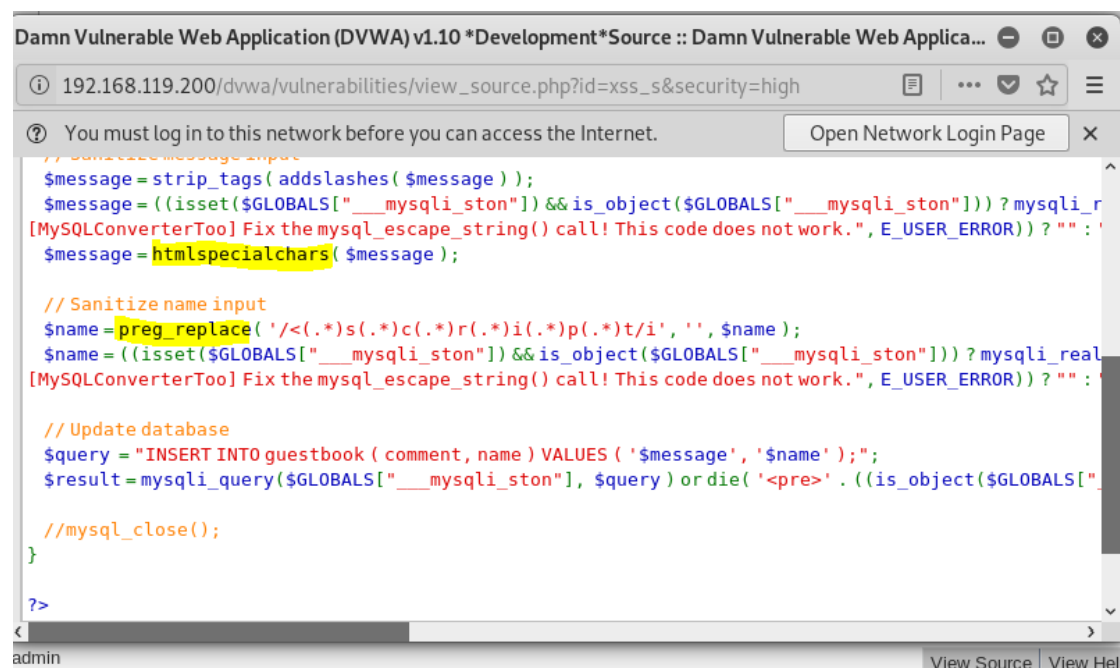


图 3.1

步骤 2: 方法与反射型 XSS 一致, 不能使用 `<script>` 标签, 但可以使用 `` 标签。由于前端仍然有字符长度限制, 所以仍需要使用 Burpsuite 来修改数据包。在 Burpsuite 中修改数据包中的 txtName 内容为 ``, 如图 3.2。放行数据包后成功弹出 cookie, 如图 3.3。

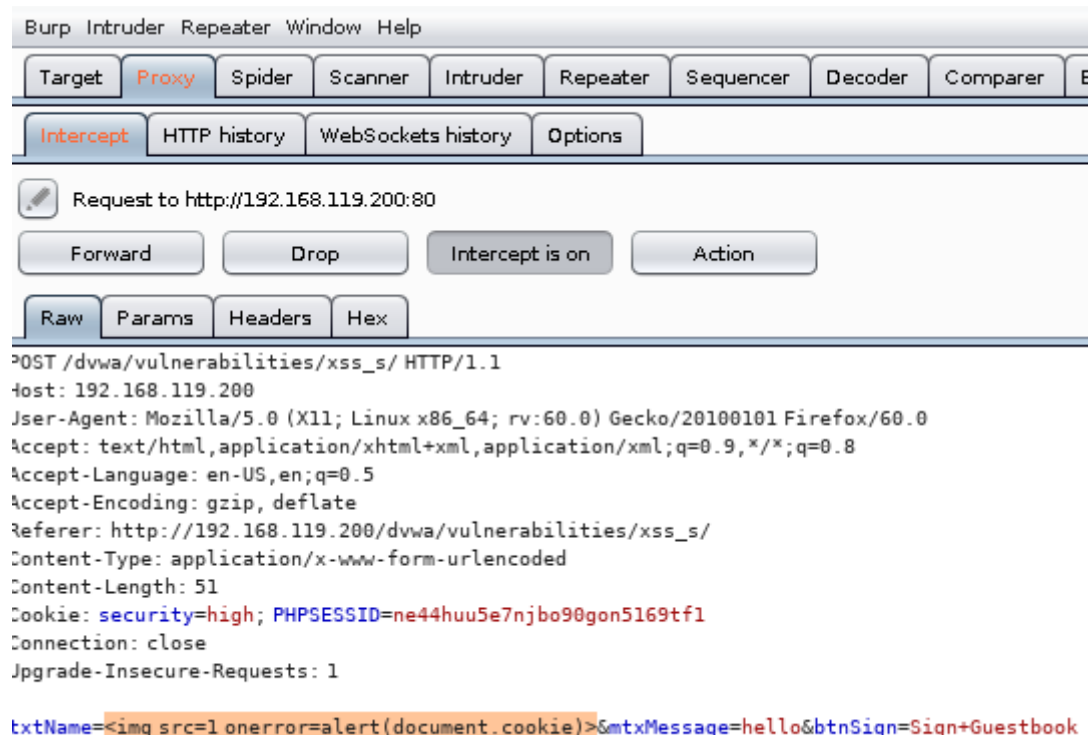


图 3.2

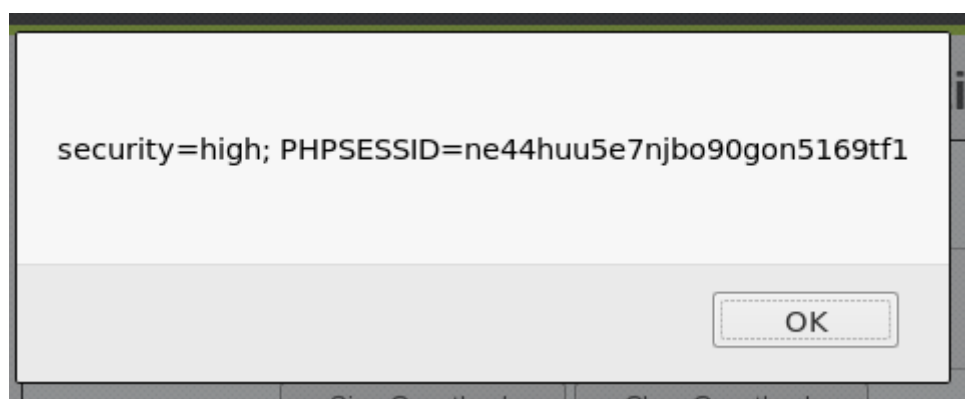


图 3.3