《安全程序设计与实践》
实验报告

实验名称：Session 欺骗攻击和 Cookie 欺骗攻击

实验类型：实操型

指导教师：孙海峰

专业班级：

姓　　名：

学　　号：

实验地点：东 11C227

实验日期：2025/5/21

一、 实验目的

1. 可以解释分析 Session 欺骗原理危害，掌握多种防护方式应用，熟练运用防护手段，增强网络安全性。
2. 可以解释 Cookie 欺骗原理及危害，应用特殊键值对和客户端信息一致性检测防护 Cookie 欺骗攻击。

二、 实验过程

Session 欺骗攻击

任务一：Session 欺骗攻击测试

1-1 任务实现

将 fuctions.php 中 check_user_agent 函数（用于验证 User-Agent 一致性）注释掉。

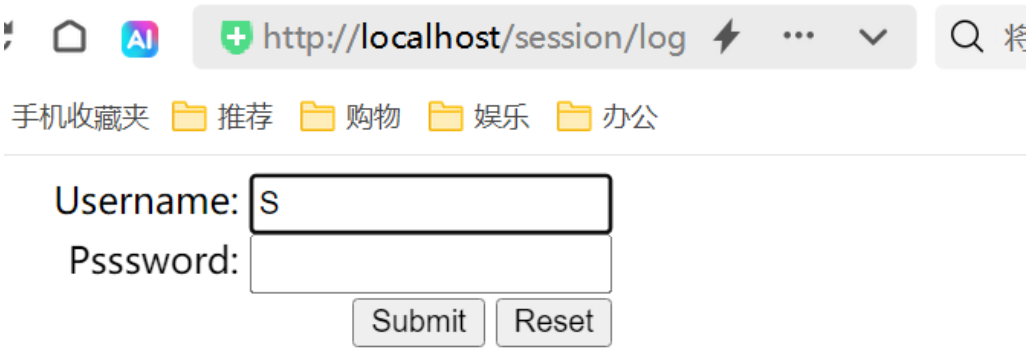
正常情况下，服务器会记录登录时的 User-Agent，并在后续请求中验证，防止跨浏览器会话劫持。

```php
function check_user_agent()//检查User-Agent的一致性
{
    if (isset($_SESSION['HTTP_USER_AGENT']))
    {
        if ($_SESSION['HTTP_USER_AGENT'] != md5($_SERVER['HTTP_USER_AGENT']))
        {
            exit('客户端信息异常');
        }
    }
    else
    {
        $_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
    }

}
```

注释后，服务器不再验证客户端环境一致性，为 Session 欺骗埋下隐患。

网页效果：



1-2 从浏览器复制 session

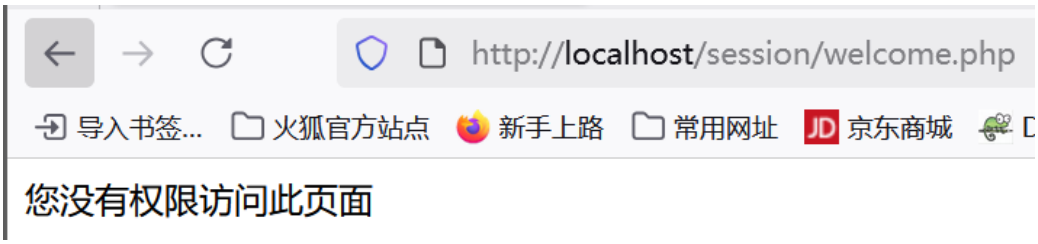
登录 admin 用户，打开浏览器开发者工具（F12），切换至 “网络” 标签页。

刷新页面，找到 check_login_mysqli.php 请求，在 “响应头” 中获取 PHPSESSID 值：p3t4p210ts63rasrlebpmhfteb。

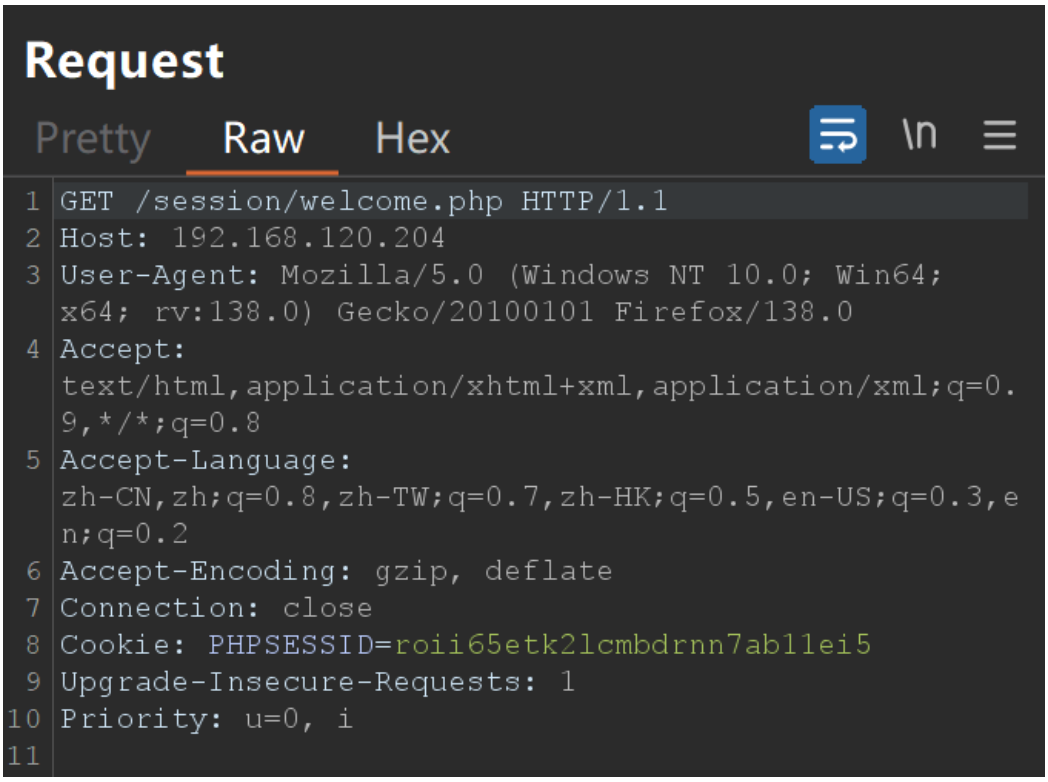
1-3 Session 欺骗攻击实施

使用 Burp Suite 或浏览器自带工具修改请求头。

访问 welcome.php，发现没用权限访问。



进行抓包，可以看到存在 PHPSESSID，我们修改为之前得到的 admin 的 SESSID（p3t4p210ts63rasrlebpmhfteb）进行重新发包。



发送请求后，服务器返回欢迎用户 admin 登陆，证明会话劫持成功。

1-4 测试分析

Session 欺骗（Session Hijacking）是指攻击者通过窃取合法用户的会话标识（如PHPSESSID），伪装成目标用户身份访问系统的攻击方式。

用户认证通过后，服务器生成唯一的 SessionID，存储用户状态（如$_SESSION['username']='admin'），并通过 Cookie 返回 PHPSESSID。

后续请求携带 PHPSESSID，服务器根据 ID 查询会话数据，无需重新认证。

若 SessionID 被窃取，攻击者可直接伪造请求，服务器误认为是合法用户。

任务二：Session 欺骗攻击防护

2-1 使用注销机制退出登录

我们将 360 浏览器直接关闭，仍然可以用 session 在火狐浏览器进行登录，因为此时浏览器只是回收了 Cookie 信息，但服务器并不知道浏览器已经关闭，只要 Session 没有超时，他在服务器中仍然存在。

但是如果在 360 浏览器中退出账户后，再去用他的 session 去进行欺骗是欺骗失败的。因为用户服务器会收到相应的请求，并执行注销操作，通常会销毁与该用户相关的 Session，攻击者原先获得的 Session 就失效了。

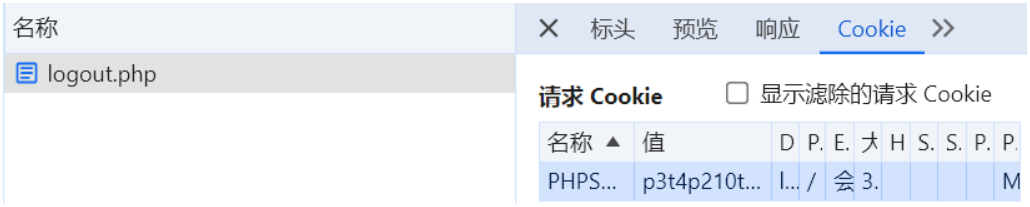
在 logout.php 中实现：

```php
<?php
session_start();
session_unset();
session_destroy();
echo "注销成功";
?>
```

直接关闭浏览器时，服务器会话未失效，攻击者仍可利用 SessionID 登录；

调用 logout.php 后，服务器会话被销毁，攻击者携带的 SessionID 失效。

与之前测试一样的方法进行测试。



发现改包重新发送之后还是显示“没有权限访问此页面”。

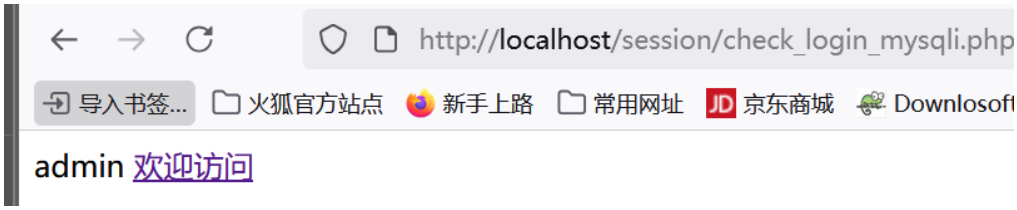
2-2 给 Sesssion 设置生存时间

服务器定期清理超过 gc_maxlifetime 的会话文件，超时后 SessionID 失效。

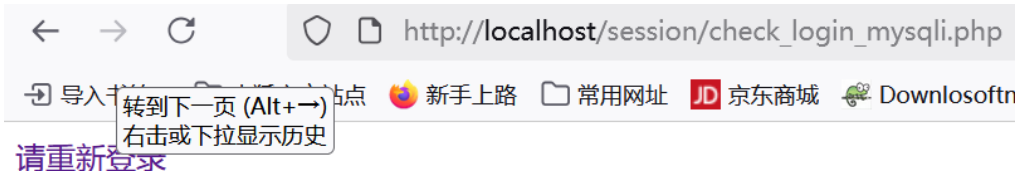
按照指导书修改 php.ini，设置生存时间为 10，重新启动 apache。

验证：

登录 admin 账户



登录成功，过 10s 后刷新出现，重新登录，说明 session 失效。



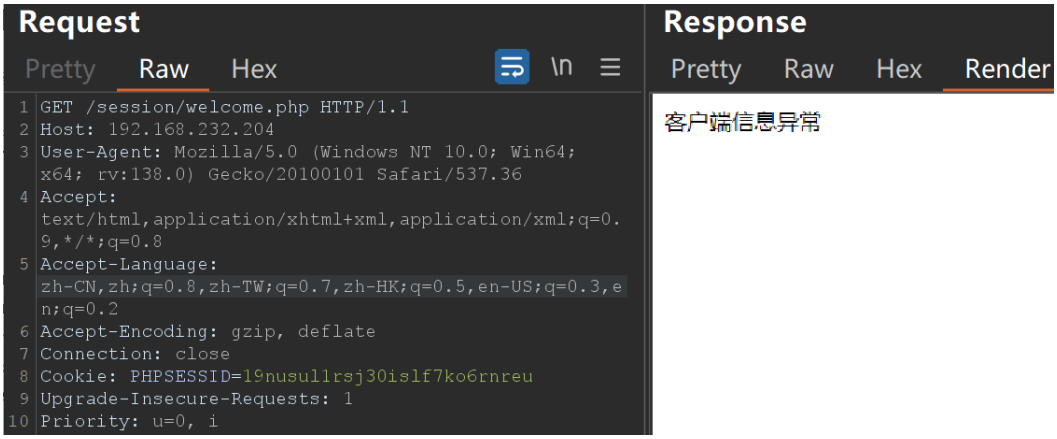
2-3 检测 User-Agent 的一致性

在 function 函数中加入函数——判断 session 数组的内容和 user-agent 的内容是否一致，如果不一致说明异常访问，我们在火狐浏览器中进行 admin 用户登录，然后进行 session 欺骗攻击。

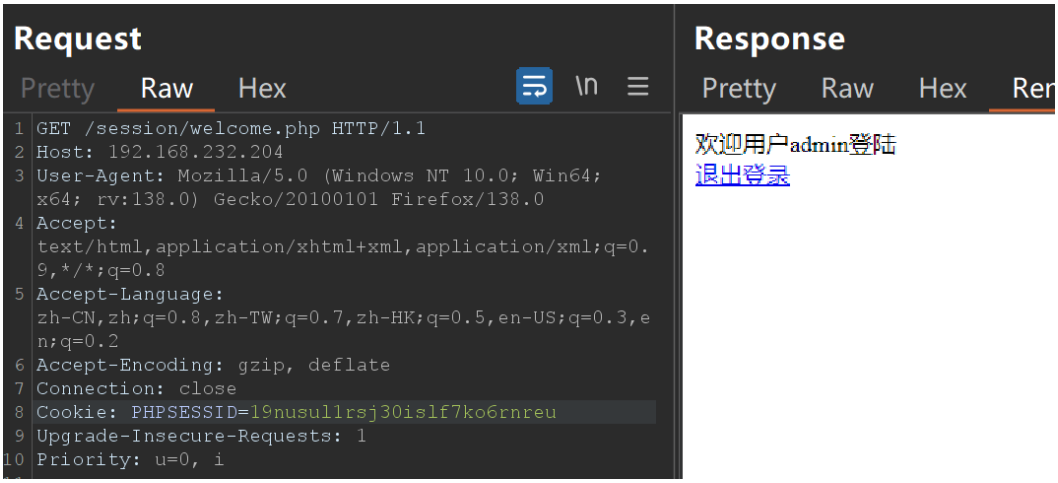
```php
function check_user_agent()//检查User-Agent的一致性
{
    if (isset($_SESSION['HTTP_USER_AGENT']))
    {
        if ($_SESSION['HTTP_USER_AGENT'] != md5($_SERVER['HTTP_USER_AGENT']))
        {
            exit('客户端信息异常');
        }
    }
    else
    {
        $_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
    }

}
```

使用 360 浏览器，修改为 admin 的 session，发现访问失败，这是由于 user-agent 不一致，服务器返回客户端信息异常。



抓包修改 cookie：phpsessid 为 admin 的 session 和 user-agent 为火狐浏览器的信息，发现欺骗成功。这是因为 user-agent 也保持了一致。



2-4 重置 SessionID

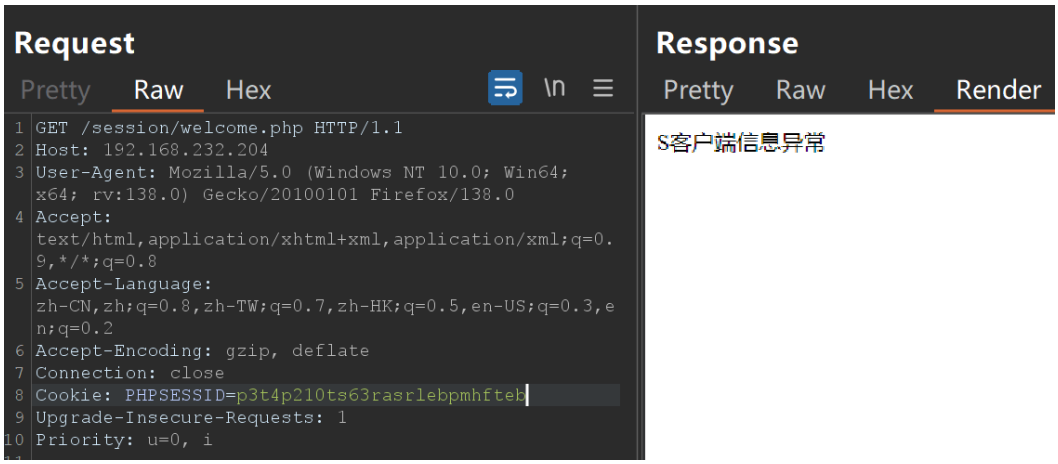
在 function.php 中开启重置 sessionID，session_regenerate_id(true);

```php
function start_session($expire = 0)
{
    session_start();
    if ($expire != 0 && isset($_SESSION['last_visit']))
    {
        $time_last = time() - $_SESSION['last_visit'];
        if ($time_last >= $expire)
        {
            session_unset();
            session_destroy();
            exit("<a href='login.html'>请重新登录</a>");
        }
    }
    $_SESSION['last_visit'] = time();
    check_user_agent();
    session_regenerate_id(true); //重置SessionID
    setcookie(session_name(), session_id(), NULL, '/', NULL, FALSE, TRUE);
}
```

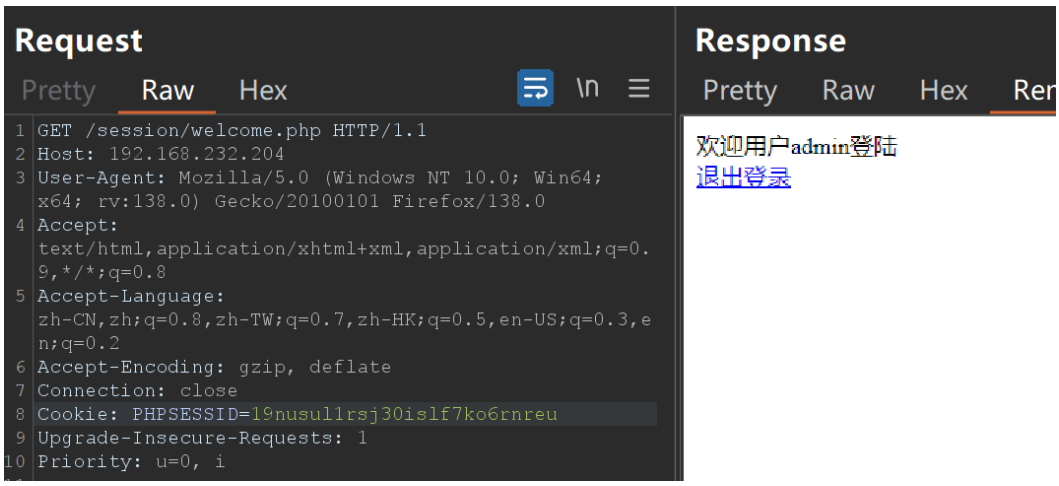
session_regenerate_id(true)会删除旧会话文件，生成新 ID，防止 Session 固定攻击，攻击者若仅获取旧 SessionID，无法通过新 ID 验证。

我们进行 Session 重置之后进行测试。

然后在 360 进行登录，登录成功后使用火狐进行欺骗，欺骗失败



重新获取 SessionID，欺骗成功。



Cookie 欺骗攻击

任务三：Cookie 欺骗攻击测试

3-1 测试准备

添加 uagent 字段

```
mysql> use lab;
Database changed
mysql> alter table users add uagent varchar(32);
Query OK, 5 rows affected (0.66 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

用于记录登录时的 User-Agent，后续验证请求合法性。

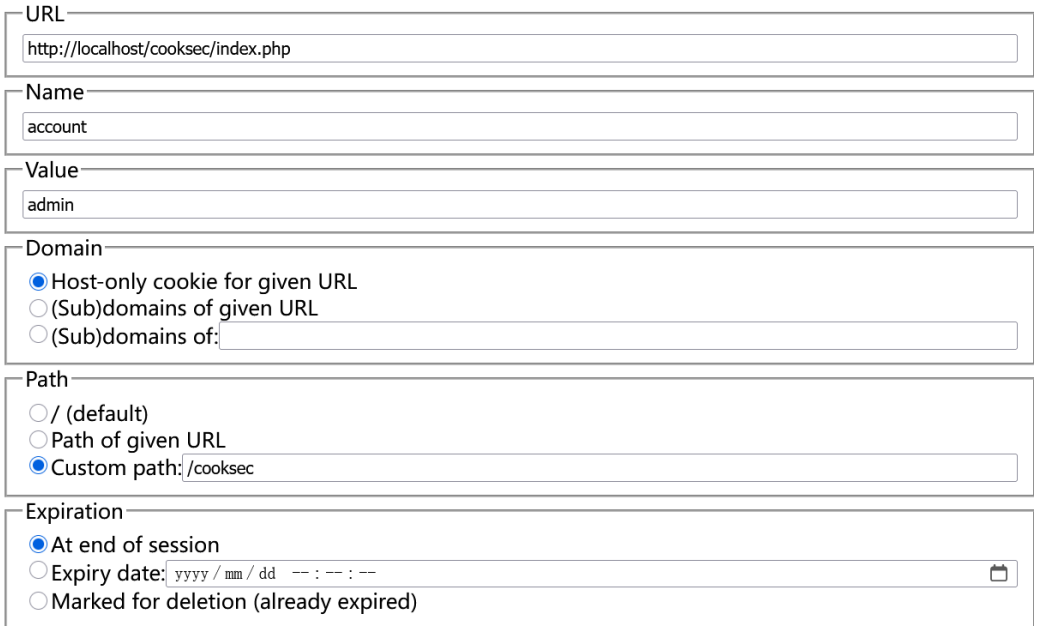
```
mysql> select * from users;
+----+----------+----------+--------+
| id | username | passcode | uagent |
+----+----------+----------+--------+
|  1 | admin    | admin123 | NULL   |
|  2 | alice    | alice456 | NULL   |
|  3 | test     | 123      | NULL   |
|  6 | admin'#  | 123456   | NULL   |
|  7 | ttt      | 111      | NULL   |
+----+----------+----------+--------+
5 rows in set (0.00 sec)
```

使用 Cookie Manager 插件清除所有 Cookie，确保测试环境干净。

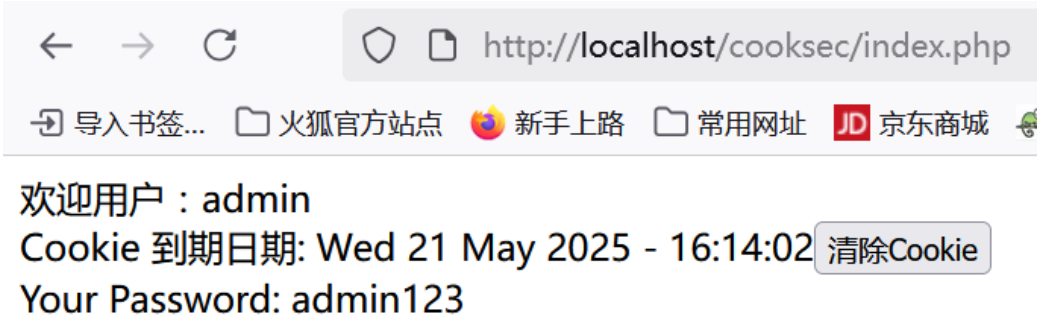
3-2 测试实施

使用 cookie manager 新增浏览器 cookie，猜测 cookie 账号：

Name: account，Value: admin，Path: /cooksec；



刷新页面后，系统显示欢迎用户:admin，证明 Cookie 欺骗成功。



3-3 测试分析

Cookie 欺骗是利用客户端存储的 Cookie 信息进行身份伪造的攻击方式。核心原理包括：

Cookie 的明文存储风险：若 Cookie 直接存储用户身份（如 account=admin），攻击者可通过猜测 Cookie 值伪造身份；

Name 为“account”表明这个 Cookie 是与账户相关的标识，Value 为“admin”代表登录账户是管理员身份，Path 选择选择“Custom path:/cooksec”，意味着它仅在“/cooksec”路径及其子路径下生效，Expiration（有效期）选的是“At end of session”，即会话结束时过期，也就是浏览器关闭时该 Cookie 失效。

这个欺骗过程是系统直接通过 accountCookie 值判断用户身份，未做任何验证。猜测 Cookie 键值对（account=admin）即可伪造管理员身份，属于“明文 Cookie 认证”漏洞。

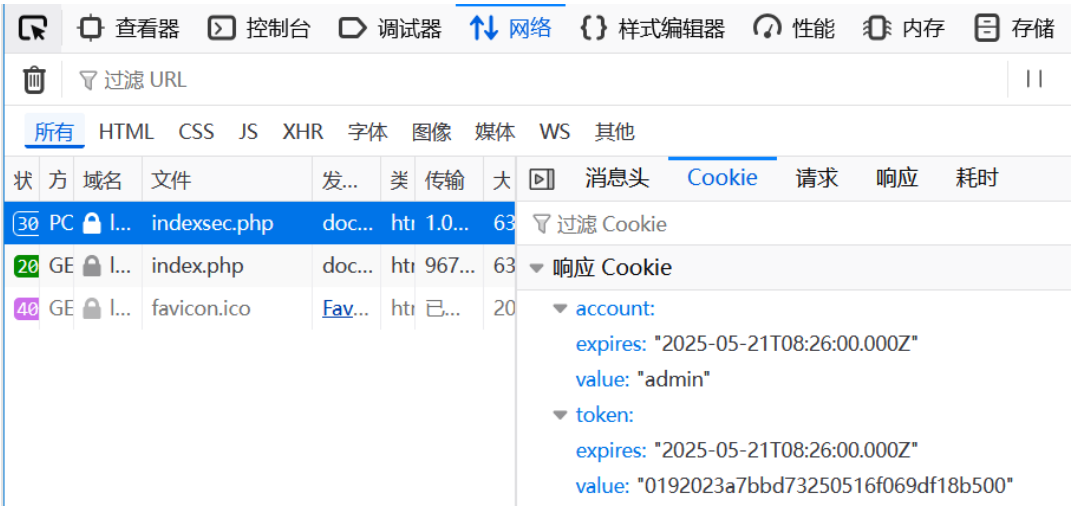
任务四：Cookie 欺骗攻击防护

4-1 设置特殊键值对

在 function 代码中添加 setcookie('token', md5($row[2]),$timestamp);

在设置 token 值是对$row[2]进行 MD5 哈希处理后的结果，并设置了过期时间为$timestamp。MD5 哈希是一种不可逆的摘要算法，理论上不同内容产生相同哈希值概率较低，用于生成一个相对唯一的标识。

token 结合密码和 User-Agent 生成，攻击者无法仅凭 account=admin 伪造有效 token，密码存储于服务器，攻击者无法获取原始值，哈希值难以逆向。



清空 cookie，进行 cookie 欺骗攻击：

仅伪造 account=admin，未伪造 token，登录失败；

发现无法登录成功，但是如果在 360 中登录拿到 account 和 token 的键值对，使用火狐进行 cookie 欺骗则成功登录。

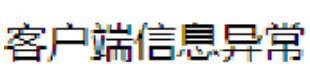
4-2 检测 User-Agent 的一致性

添加 user-agent 的验证机制，user - agent 包含了浏览器类型、版本、操作系统等信息，服务器可以借此识别客户端使用的浏览器等环境信息

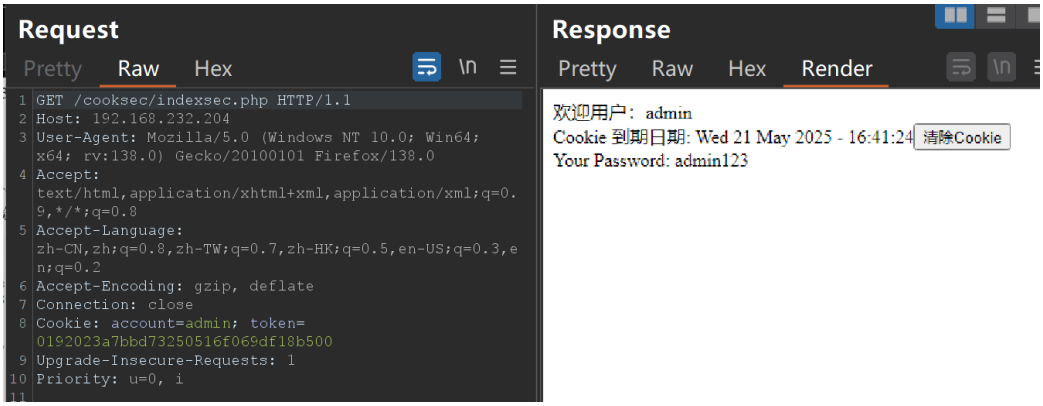
```php
if($row[0])
{
    $update = "update users set uagent=? where id=$row[0]";
    $stmt = $con->prepare($update);
    if ($stmt)
    {
        $stmt->bind_param("s", md5($_SERVER['HTTP_USER_AGENT']));
        $stmt->execute();
    }
    $stmt->close();

    $timestamp = time() + $exipre_time;
    setcookie('account', $row[1], $timestamp);
    setcookie('token', md5($row[2]),$timestamp);
    header ('Location: index.php');
}
else
    echo "<script>alert('用户名或密码错误!'); history.go(-1);</script>";
```

然后和之前一样进行 cookie 猜测欺骗，发现登陆失败。



同时伪造 User-Agent 和 Cookie，验证通过，证明单一 User-Agent 验证仍可被抓包篡改。



三、 实验结论

Session/Cookie 欺骗攻击依赖会话标识的窃取或伪造，核心在于身份验证的单一性，即仅通过标识验证用户身份。

标识安全方面需实现 SessionID 随机化、Cookie 加密签名并设置 HttpOnly/Secure 等属性，传输安全依赖 HTTPS 加密防止标识传输中被嗅探，验证增强可结合 User-Agent 与浏览器指纹、多因素认证及行为分析，生命周期管理则包括会话超时、注销销毁会话及定期更换标识

然而技术防护存在固有局限，如无法抵御社会工程学窃取标识或零日漏洞攻击，因此需结合网络层和应用层形成纵深防御，避免单一策略被绕过。

同时通过算法升级、浏览器特性利及代码层安全设计持续提升防护能力，最终从技术、流程、人员多维度构建动态防御体系以应对攻击。

四、 思考题

1. 如果攻击者可以直接从登录用户的浏览器拿走 SessionID 等信息，是否可以通过采用技术手段杜绝 Session 欺骗？为什么？

   只能通过多层防护增加攻击成本，无法完全杜绝：

   - 攻击者可通过 XSS、钓鱼邮件、公共 WiFi 嗅探等手段获取 SessionID，技术防护无法覆盖所有社会工程学攻击；
   - 零日漏洞可能绕过现有防护（如服务器未打补丁的 Session 处理漏洞）。
   - 注销机制：攻击者可在用户注销前滥用 SessionID；
   - 超时设置：若攻击在有效期内发起，仍可成功；
   - User-Agent 验证：抓包工具可伪造任意请求头，绕过验证。

2. 浏览器的请求头部的 Referer 信息是否适合 Session 欺骗的防护？为什么？

   不合适。

   - Referer 的不可靠性：浏览器可手动禁用 Referer 发送，或因隐私设置自动省略；跨域请求时 Referer 可能被修改或省略，无法作为信任依据。
   - 伪造成本低：攻击者可通过抓包工具任意修改 Referer 值，如设置为合法域名；Referer 主要用于防御 CSRF，而非会话劫持，场景不匹配。

   使用 SameSiteCookie 属性防止跨站请求伪造，比 Referer 更可靠。

3. 如果攻击者可以直接从登录用户的浏览器拿走 Cookie 信息，是否可以通过采用技

术手段杜绝 Cookie 欺骗？为什么？

不能杜绝，只能增加 Cookie 欺骗难度：

- 客户端存储本质风险：Cookie 存储于客户端，攻击者可通过本地恶意程序（如键盘记录器）或浏览器漏洞直接读取；使加密 Cookie，密钥若泄露或算法被破解，仍可被伪造。
- 验证机制的脆弱性：依赖 User-Agent 等可伪造的请求头，无法构建绝对信任链；多设备登录场景下，强制验证 User-Agent 会影响用户体验。

结合 Cookie 加密、服务器端会话绑定、行为分析构建动态防护，而非单一技术。

4. 浏览器的请求头部的 Referer 信息是否适合 Cookie 欺骗的防护？为什么？

不适合，Referer 信息不能有效防护 Cookie 欺骗，Referer 信息主要用于防止跨站请求伪造（CSRF）攻击，它容易被篡改，且在某些情况下可能不可用。