



西安电子科技大学
XIDIAN UNIVERSITY

Web 安全

课程作业

作业内容:

小组作业 2

小组成员:

姜欣悦

李颖

盖乐

职泉

陈静怡

实验日期:

2023.12.28

目录

SQL 注入实验	4
一、 攻击方式简介	4
二、 实验目标	4
三、 实验步骤	4
四、 实验过程	5
五、 实验结果与防御分析	8
命令注入实验	8
一、 攻击方式简介	8
二、 实验目标	8
三、 实验步骤	9
四、 实验过程	9
XSS 注入实验	11
一、 攻击方式简介	11
二、 实验目标	11
三、 实验步骤	11
四、 实验过程	12
五、 实验结果与防御分析	14
CSRF 注入实验	14
一、 攻击方式简介	14
二、 实验目标	15
三、 实验步骤	15

四、 实验过程	16
五、 实验结果与防御分析	17
作业 2 小组分工	21

SQL 注入实验

一、 攻击方式简介

SQL 是操作数据库数据的结构化查询语言，网页的应用数据和后台数据库中的数据在进行交互时会采用 SQL。而 SQL 注入是将 Web 页面的原 URL、表单域或数据包输入的参数，修改拼接成 SQL 语句，传递给 Web 服务器，进而传给数据库服务器以执行数据库命令。如 Web 应用程序的开发人员对用户所输入的数据或 cookie 等内容不进行过滤或验证(即存在注入点)就直接传输给数据库，就可能导致拼接的 SQL 被执行，获取对数据库的信息以及提权，发生 SQL 注入攻击。

二、 实验目标

实验基于已有的 Web 应用平台进行攻击测试，实现 SQL 的简单注入，并对攻击过程和结果进行分析。深刻理解 Web 安全在 Web 应用中的重要性、关键性。深刻理解 Web 应用开发过程中需要注意的 SQL 数据库接口的安全的重要性。

三、 实验步骤

第一步：SQL 注入点探测。探测 SQL 注入点是关键的一步，通过适当的分析应用程序，可以判断什么地方存在 SQL 注入点。

通常只要带有输入提交的动态网页，并且动态网页访问数据库，就可能存在 SQL 注入漏洞。如果程序员信息安全意识不强，采用动态构造 SQL 语句访问数据库，并且对用户的输入未进行有效性验证，则存在 SQL 注入漏洞的可能性很

大。一般通过页面的报错信息来确定是否存在 SQL 注入漏洞。

第二步：收集后台数据库信息。不同数据库的注入方法、函数都不尽相同，因此在注入之前，我们先要判断一下数据库的类型。判断数据库类型的方法很多，可以输入特殊字符，如单引号，让程序返回错误信息，我们根据错误信息提示进行判断；还可以使用特定函数来判断，比如输入“1 and version（）>0”，程序返回正常，说明 version（）函数被数据库识别并执行，而 version（）函数是 MySQL 特有的函数，因此可以推断后台数据库为 MySQL。

第三步：猜解用户名和密码。数据库中的表和字段命名一般都是有规律的。通过构造特殊 SQL 语句在数据库中依次猜解出表名、字段名、字段数、用户名和密码。

第四步：查找 Web 后台管理入口。WEB 后台管理通常不对普通用户开放，要找到后台管理的登录网址，可以利用 Web 目录扫描工具（如：wwwscan、AWVS）快速搜索到可能的登录地址，然后逐一尝试，便可以找到后台管理平台的登录网址。

第五步：入侵和破坏。一般后台管理具有较高权限和较多的功能，使用前面已破译的用户名、密码成功登录后台管理平台后，就可以任意进行破坏，比如上传木马、篡改网页、修改和窃取信息等，还可以进一步提权，入侵 Web 服务器和数据库服务器。

四、 实验过程

web 软件 SQL 注入攻击

在制作 web 应用时，为方便实验测试，将登录页面的 email 和 password 进行字符串传输，以及 SQL 的直接查找。这种方式具有 SQL 安全漏洞，后面将进行 SQL 攻击测试。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.demo.mapper.UserMapper">

  <select id="queryUserList" resultType="user" parameterMap="user">
    select * from uav.uavusers where email = uavusers.email and password = uavusers.password
  </select>

</mapper>
```

图 1 后端查阅有无用户所使用的 SQL 语句

首先，页面对输入内容进行了一定的判断，进行内容安全保护。当输入不像电子邮件形式的文字进去后，会检测出异常，因此我们需要进行伪造。

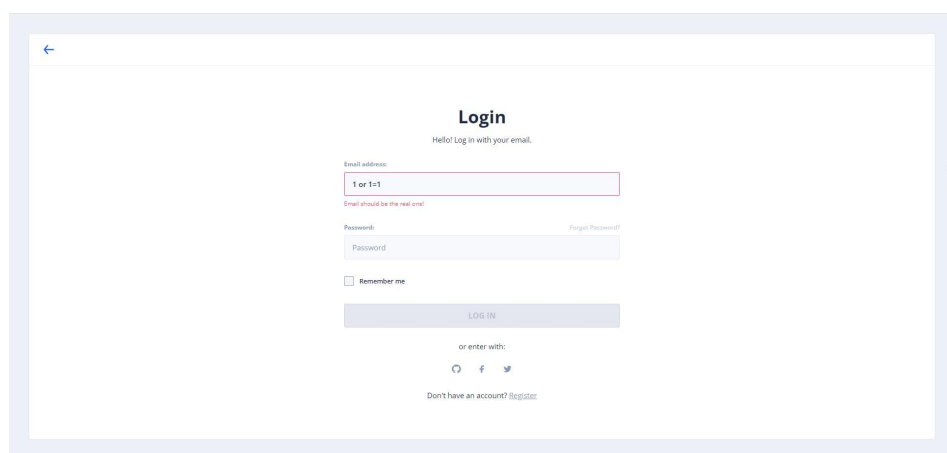
A screenshot of a web application's login page. The page has a light blue header and a white main content area. The title "Login" is centered at the top. Below the title, there is a sub-header "Hello! Log in with your email." followed by two input fields: "Email address" and "Password". The "Email address" field contains the text "1 or 1=1". Below the "Email address" field, there is a small red error message that says "Email should be the real email!". The "Password" field is empty. Below the "Password" field, there is a checkbox labeled "Remember me" which is unchecked. At the bottom of the form, there is a blue "LOG IN" button. Below the button, there is a link "or enter with:" followed by social media icons for GitHub, Facebook, and Twitter. At the very bottom, there is a link "Don't have an account? Register".

图 2 登陆页面检测出输入异常

我们对 SQL 判断语句绕过，并伪造成类似电子邮件的样子，成功输入。

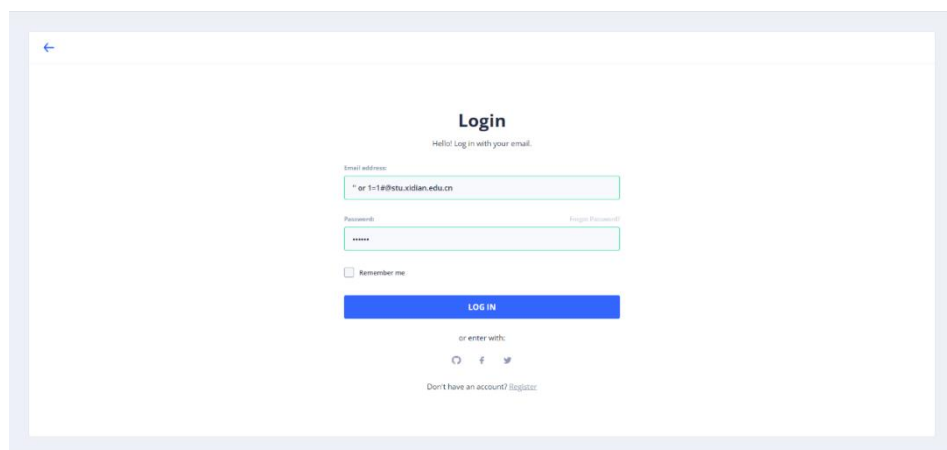
A screenshot of the same web application's login page. The "Email address" field now contains the text "1-or-1@stu.xidian.edu.cn". The "Password" field is still empty. The "Remember me" checkbox is still unchecked. The "LOG IN" button is now highlighted in blue. The error message "Email should be the real email!" is no longer visible. The rest of the page layout remains the same.

图 3 伪造语句输入

输入以后，前端通过 http 协议发出 GET 请求向后端请求数据。

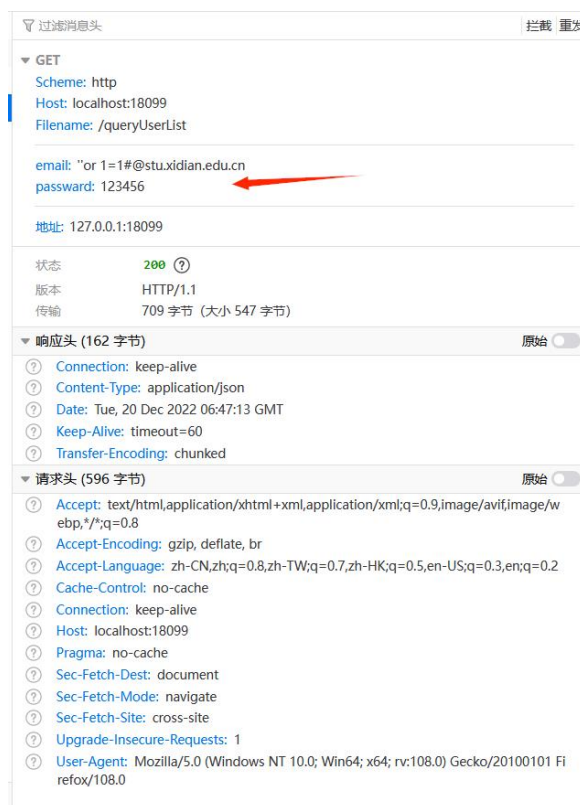


图 4 通过抓包分析 http 协议 GET 请求

后端接收到请求后，运行 SQL 方法，但是由于前面巧妙地构造，SQL 语句会绕过原本的判断。

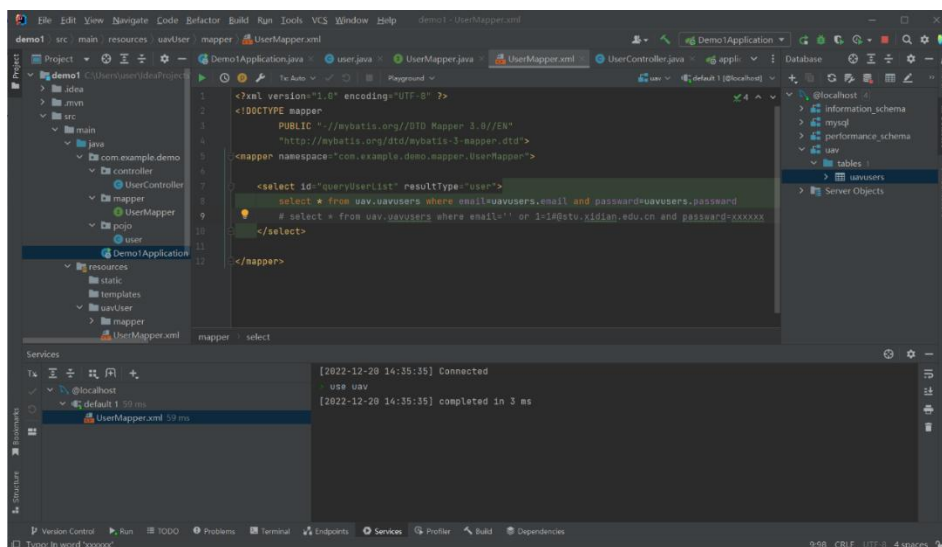


图 5 后端接收并运行 SQL 方法

在判断语句被绕过，将会反馈数据库的内容，前端将会返回并展示出信息。

```

  "id": 1
},
{
  "name": "harrywang",
  "email": "1191477582@qq.com",
  "password": "123456",
  "id": 2
},
{
  "name": "Nick Jones",
  "email": "1191477582@qq.com",
  "password": "123456",
  "id": 3
},
{
  "name": "Eva Rose",
  "email": "1191477582@qq.com",
  "password": "123456",
  "id": 4
},
{
  "name": "Jack Williams",
  "email": "1191477582@qq.com",
  "password": "123456",
  "id": 5
},
{
  "name": "Lee Wong",
  "email": "1191477582@qq.com",
  "password": "123456",
  "id": 6
},
{
  "name": "Alan Thompson",
  "email": "1191477582@qq.com",
  "password": "123456",
  "id": 7
},
{
  "name": "Kate Martinez",
  "email": "1191477582@qq.com",
  "id": 8
}
]

```

图 6 Firefox 浏览器解析数据库信息后的前端页面

五、实验结果与防御分析

数据库内容被显示在浏览器页面，这次攻击成功的主要原因在于明文传参。修改为加密密文传递后，解决问题。

命令注入实验

一、攻击方式简介

命令注入攻击的常见模式为：仅仅需要输入数据的场合，却伴随着数据同时输入了恶意代码，而装载数据的系统对此并未设计良好的过滤过程，导致恶意代码也一并执行，最终导致信息泄露或者正常数据的破坏。其中最常见的一类攻击行为就是针对数据库系统的 SQL（结构化查询语言）注入。常见的 SQL 语句构成为：对符合特定条件的数据（某些行的某些列）实施增、删、改、查等操作。其中需要符合的条件就是所谓“数据”（如学生数据表中性别为女，或年龄大于 10 岁等），对这些数据的选取以及实施的某种操作就是“指令”。

二、实验目标

实验基于已有的 Web 应用平台进行攻击测试，实现简单的命令注入，并对攻

击过程和结果进行分析。深刻理解 Web 安全在 Web 应用中的重要性、关键性。

深刻理解 Web 应用开发过程中需要注意的命令注入安全的重要性。

三、 实验步骤

1. 寻找程序中是否调用了系统命令
2. 系统命令或程序函数的参数是否可控、可改
3. 是否有接口可以注入命令
4. 注入并破坏内容

四、 实验过程

命令注入实验

与上面对登录页面的 SQL 攻击类似，对注册网页进行 SQL 攻击。

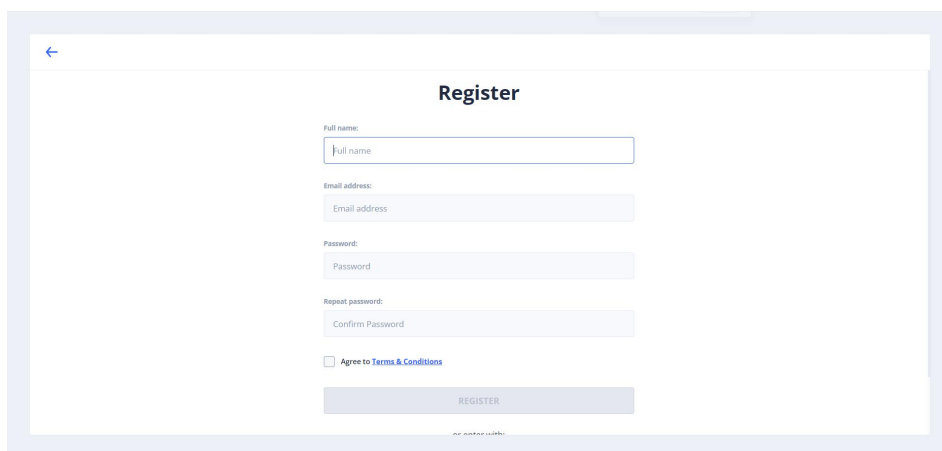
A screenshot of a web registration form titled "Register". The form is centered on a light gray background. It contains several input fields: "Full name:" with a placeholder "Full name", "Email address:" with a placeholder "Email address", "Password:" with a placeholder "Password", and "Repeat password:" with a placeholder "Confirm Password". Below these fields is a checkbox labeled "Agree to Terms & Conditions". At the bottom of the form is a gray button labeled "REGISTER". The entire form is enclosed in a light blue border.

图 7 注册页面

注册页面使用的 SQL 语句为：

```
ALTER TABLE table_name  
ADD [COLUMN] column_definition;
```

可以通过注册页面的窗口，添加非法信息进入数据库。

	name	email	password	id	picture	contacts	recentUsers
X	haoyuwang	119147758;	123456789		1 assets/imag	(Null)	(Null)
	Nick Jones	119147758;	123456		2 assets/imag	(Null)	(Null)
	Eva Moor	119147758;	123456		3 assets/imag	(Null)	(Null)
	Jack William	119147758;	123465		4 assets/imag	(Null)	(Null)
	Lee Wong	119147758;	123456		5 assets/imag	(Null)	(Null)
	Alan Thomp	119147758;	123456		6 assets/imag	(Null)	(Null)
	Kate Martin	119147758;	123456		7 assets/imag	(Null)	(Null)

图 8 数据库表单

在对传递参数更改为加密后的信息避免明文传输后，该攻击被解决。

XSS 注入实验

一、 攻击方式简介

XSS 攻击通常指的是通过利用网页开发时留下的漏洞，通过巧妙的方法注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。这些恶意网页程序通常是 JavaScript，但实际上也可以包括 Java、VBScript、ActiveX、Flash 或者甚至是普通的 HTML。攻击成功后，攻击者可能得到包括但不限于更高的权限（如执行一些操作）、私密网页内容、会话和 cookie 等各种内容。由于我们网页构建时功能不够齐全，很难复现 XSS 和 CSRF 两个漏洞，所以在此我们以本地搭建的 Pikachu 漏洞平台中的 XSS 和 CSRF 为例。

二、 实验目标

实验基于已有的 Web 应用平台进行攻击测试，实现简单的 XSS 注入，并对攻击过程和结果进行分析。深刻理解 Web 安全在 Web 应用中的重要性、关键性。深刻理解 Web 应用开发过程中需要注意的 XSS 注入安全的重要性。

三、 实验步骤

- 1、使用反射型 XSS 盗用 cookie，获取敏感信息。
- 2、使用存储型 XSS，植入 payload。
- 3、恶意利用结果，分析实验结果。

四、实验过程

反射型 XSS

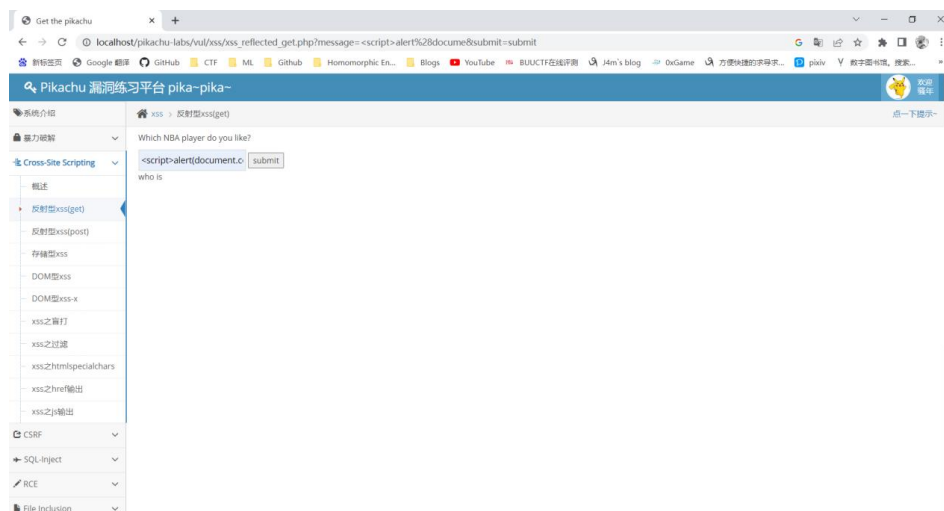


图 9 XSS 实验页面

Payload: `<script>alert(document.cookie)</script>`

该 Payload 的作用是像网页 HTML 中加入一个可以返回网页 cookie 的 JS 代码。

具体呈现出的效果是，可以观察到我们成功获取了浏览器中的 cookie。

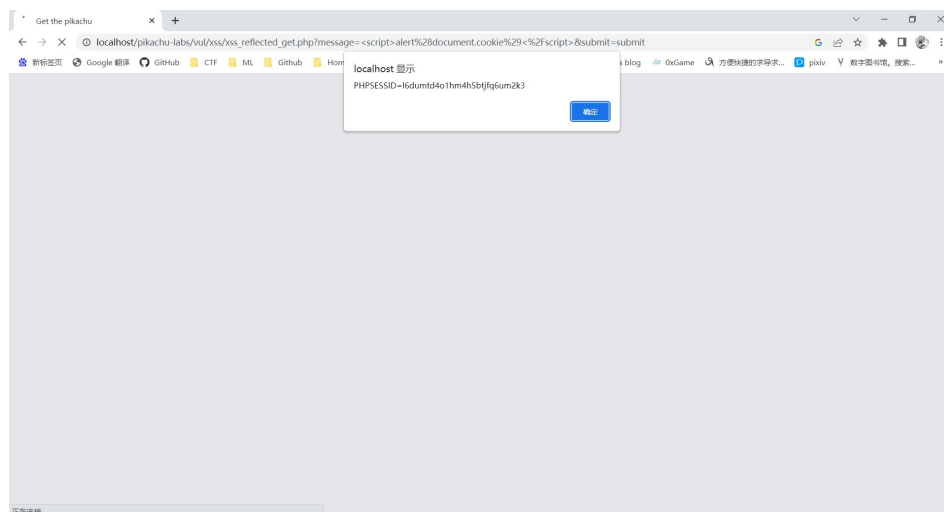


图 10 反射型 XSS 实验效果

打开 HTML 网页代码，可以观察到我们的恶意代码成功插入到了其中。

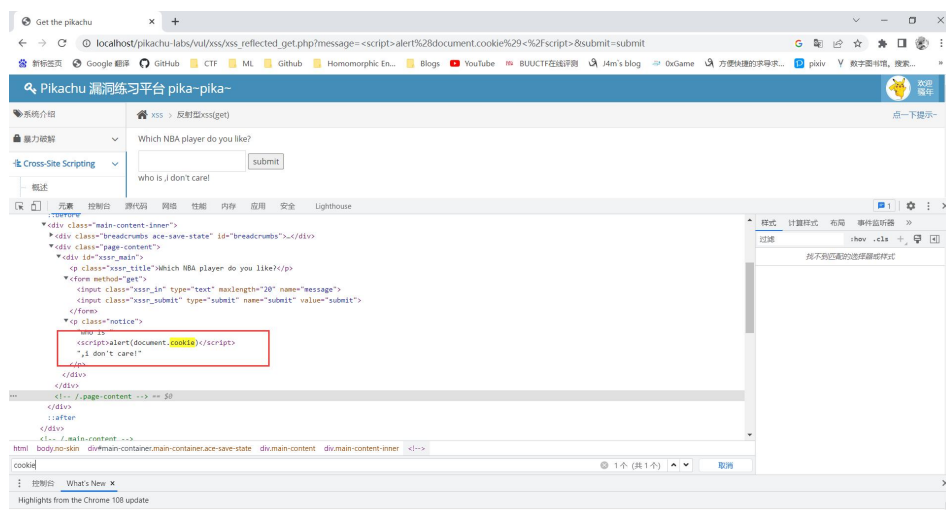


图 11 实验效果

存储型 XSS

观察到一个留言板，说明用户留下的留言会永久保存在其中，为存储型 XSS。

留下恶意 Payload。

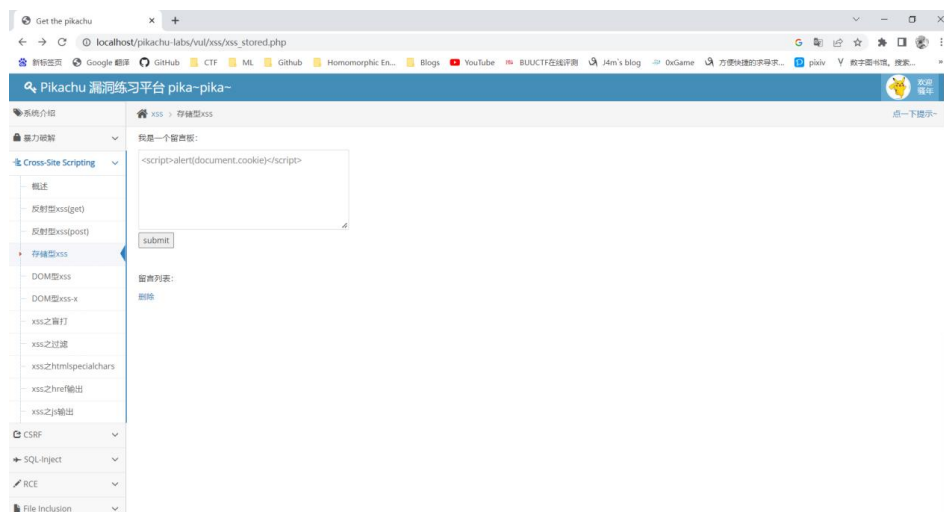


图 12 存储型 XSS 实验平台

可以观察到，每当打开该网页的时候，都会弹出 cookie，从而证实了是存储型 XSS。

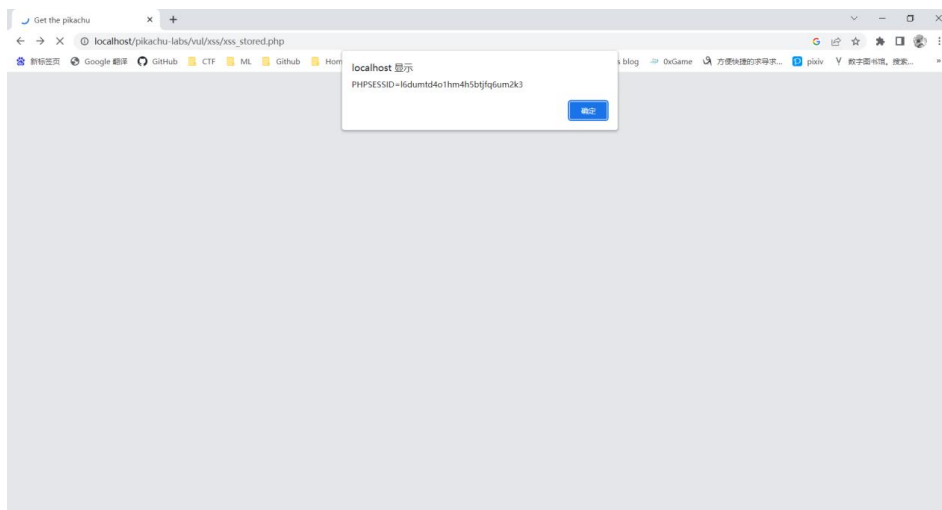


图 13 实验效果

五、 实验结果与防御分析

XSS 漏洞防御：

1、对输入和 URL 参数进行过滤(白名单和黑名单)

检查用户输入的数据中是否包含一些特殊字符，如<、>、'、"等，发现存在特殊字符，将这些特殊字符过滤或者编码。

2、HTML 实体编码

字符串 js 编码转换成实体 html 编码的方法（防范 XSS 攻击）

3、对输出内容进行编码

在变量输出到 HTML 页面时，可以使用编码或转义的方式来防御 XSS 攻击。

CSRF 注入实验

一、 攻击方式简介

跨站请求伪造（英语：Cross-site request forgery），也被称为 one-click attack

或者 session riding，通常缩写为 CSRF 或者 XSRF，是一种挟制用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。跟跨网站脚本（XSS）相比，XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户网页浏览器的信任。

二、 实验目标

实验基于已有的 Web 应用平台进行攻击测试，实现简单的 CSRF 注入，并对攻击过程和结果进行分析。深刻理解 Web 安全在 Web 应用中的重要性、关键性。深刻理解 Web 应用开发过程中需要注意的 CSRF 注入安全的重要性。

三、 实验步骤

1. 用户 C 打开浏览器，访问受信任网站 A，输入用户名和密码请求登录网站 A；
2. 在用户信息通过验证后，网站 A 产生 Cookie 信息并返回给浏览器，此时用户登录网站 A 成功，可以正常发送请求到网站 A；

用户未退出网站 A 之前，在同一浏览器中，打开一个 TAB 页访问网站 B；

网站 B 接收到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点 A；
3. 浏览器在接收到这些攻击性代码后，根据网站 B 的请求，在用户不知情的情况下携带 Cookie 信息，向网站 A 发出请求。网站 A 并不知道该请求其实是由 B 发起的，所以会根据用户 C 的 Cookie 信息以 C 的权限处理该请求，导致来自网站 B 的恶意代码被执行。

四、实验过程

CSRF 注入实验

首先使用 kobe 的身份进行登录。



图 14 伪造登录

准备进行个人信息的修改

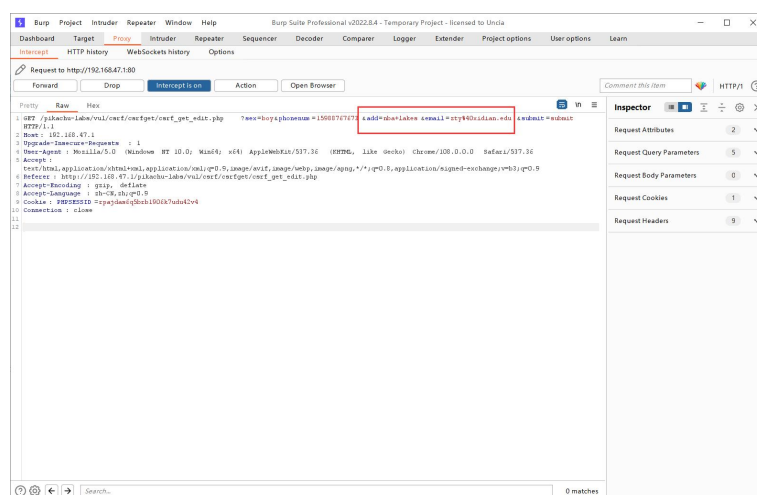


图 15 修改个人信息

抓包后发现请求头没有 Refer 参数，基本可以判断存在 CSRF 注入漏洞，因此复制 URL 到浏览器中，修改 add 为 China，email 为 CSRF@xidian.edu/pikachu-labs/vul/csrf/csrfget/csrf_get_edit.php?sex=boy&phonenumber=15988767673&add=China&email=CSRF@xidian.edu&submit=submit 可以看到修改成功。



图 16 查看修改结果

五、 实验结果与防御分析

CSRF 的防御：

1.验证 HTTP Referer 字段

根据 HTTP 协议，在 HTTP 头中有一个字段叫 Referer，它记录了该 HTTP 请求的来源地址。在通常情况下，访问一个安全受限页面的请求来自于同一个网站，比如需要访问：

http://bank.example/withdraw?account=bob&amount=1000000&for=Mallory，

用户必须先登陆 bank.example，然后通过点击页面上的按钮来触发转账事件。

这时，该转账请求的 Referer 值就会是转账按钮所在的页面的 URL，通常是以 bank.example 域名开头的地址。而如果黑客要对银行网站实施 CSRF 攻击，他只能在他自己的网站构造请求，当用户通过黑客的网站发送请求到银行时，该请求的 Referer 是指向黑客自己的网站。因此，要防御 CSRF 攻击，银行网站只需要对于每一个转账请求验证其 Referer 值，如果是以 bank.example 开头的域名，则说明该请求是来自银行网站自己的请求，是合法的。如果 Referer 是其他网站的话，则有可能是黑客的 CSRF 攻击，拒绝该请求。

这种方法的显而易见的好处就是简单易行，网站的普通开发人员不需要操心 CSRF 的漏洞，只需要在最后给所有安全敏感的请求统一增加一个拦截器来检查 Referer 的值就可以。特别是对于当前现有的系统，不需要改变当前系统的任何已有代码和逻辑，没有风险，非常便捷

然而，这种方法并非万无一失。Referer 的值是由浏览器提供的，虽然 HTTP 协议上有明确的要求，但是每个浏览器对于 Referer 的具体实现可能有差别，并不能保证浏览器自身没有安全漏洞。使用验证 Referer 值的方法，就是把安全性都依赖于第三方（即浏览器）来保障，从理论上来讲，这样并不安全。事实上，对于某些浏览器，比如 IE6 或 FF2，目前已经有一些方法可以篡改 Referer 值。如果 bank.example 网站支持 IE6 浏览器，黑客完全可以把用户浏览器的 Referer 值设为以 bank.example 域名开头的地址，这样就可以通过验证，从而进行 CSRF 攻击。即便是使用最新的浏览器，黑客无法篡改 Referer 值，这种方法仍然有问题。因为 Referer 值会记录下用户的访问来源，有些用户认为这样会侵犯到他们自己的隐私权，特别是有些组织担心 Referer 值会把组织内网中的某些信息泄露到外网中。因此，用户自己可以设置浏览器使其在发送请求时不再提供 Referer。当他们正常访问银行网站时，网站会因为请求没有 Referer 值而认为是 CSRF 攻击，拒绝合法用户的访问

2.在请求地址中添加 token 并验证(Anti-CSRF token)

CSRF 攻击之所以能够成功，是因为黑客可以完全伪造用户的请求，该请求中所有的用户验证信息都是存在于 cookie 中，因此黑客可以在不知道这些验证信息的情况下直接利用用户自己的 cookie 来通过安全验证。要抵御 CSRF，关键在于在请求中放入黑客所不能伪造的信息，并且该信息不存在于 cookie 之中。可以在 HTTP 请求中以参数的形式加入一个随机产生的 token，并在服务

器端建立一个拦截器来验证这个 token，如果请求中没有 token 或者 token 内容不正确，则认为可能是 CSRF 攻击而拒绝该请求

这种方法要比检查 Referer 要安全一些，token 可以在用户登陆后产生并放于 session 之中，然后在每次请求时把 token 从 session 中拿出，与请求中的 token 进行比对，但这种方法的难点在于如何把 token 以参数的形式加入请求。对于 GET 请求，token 将附在请求地址之后，这样 URL 就变成 `http://url?csrftoken=tokenvalue`。而对于 POST 请求来说，要在 form 的最后加上，这样就把 token 以参数的形式加入请求了。但是，在一个网站中，可以接受请求的地方非常多，要对于每一个请求都加上 token 是很麻烦的，并且很容易漏掉，通常使用的方法就是在每次页面加载时，使用 javascript 遍历整个 dom 树，对于 dom 中所有的 a 和 form 标签后加入 token。这样可以解决大部分的请求，但是对于在页面加载之后动态生成的 html 代码，这种方法就没有作用，还需要程序员在编码时手动添加 token

该方法还有一个缺点是难以保证 token 本身的安全。特别是在一些论坛之类支持用户自己发表内容的网站，黑客可以在上面发布自己个人网站的地址。由于系统也会在这个地址后面加上 token，黑客可以在自己的网站上得到这个 token，并马上就可以发动 CSRF 攻击。为了避免这一点，系统可以在添加 token 的时候增加一个判断，如果这个链接是链到自己本站的，就在后面添加 token，如果是通向外网则不加。不过，即使这个 csrftoken 不以参数的形式附加在请求之中，黑客的网站也同样可以通过 Referer 来得到这个 token 值以发动 CSRF 攻击。这也是一些用户喜欢手动关闭浏览器 Referer 功能的原因

3.在 HTTP 头中自定义属性并验证

这种方法也是使用 token 并进行验证，和上一种方法不同的是，这里并不是把 token 以参数的形式置于 HTTP 请求之中，而是把它放到 HTTP 头中自定义的属性里。通过 XMLHttpRequest 这个类，可以一次性给所有该类请求加上 CSRFToken 这个 HTTP 头属性，并把 token 值放入其中。这样解决了上种方法在请求中加入 token 的不便，同时，通过 XMLHttpRequest 请求的地址不会被记录到浏览器的地址栏，也不用担心 token 会透过 Referer 泄露到其他网站中去

然而这种方法的局限性非常大。XMLHttpRequest 请求通常用于 Ajax 方法中对于页面局部的异步刷新，并非所有的请求都适合用这个类来发起，而且通过该类请求得到的页面不能被浏览器所记录下，从而进行前进，后退，刷新，收藏等操作，给用户带来不便。另外，对于没有进行 CSRF 防护的遗留系统来说，要采用这种方法来进行防护，要把所有请求都改为 XMLHttpRequest 请求，这样几乎是要重写整个网站，这代价无疑是不能接受的。

作业 2 小组分工

姜欣悦 20%	SQL 注入、命令注入攻击
盖乐 20%	XSS 与 CSRF 攻击
李颖 20%	Web 安全测试
陈静怡 20%	信息收集与查找 作业版式制作
职泉 20%	Web 安全测试