《信息安全综合实践》实验报告

实验名称：代码审计实验

姓名：黎锦灏 学号：518021910771 邮箱：1197993966@qq.com 实验时长：75 分钟

一、实验目的

1. 了解代码审计的原理，RCE、任意文件上传、变量覆盖等漏洞产生的原因及利用方式。
2. 尝试在代码审计过程中对特定函数进行简单利用。
3. 熟悉使用 Seay 源码进行代码审计，找到底层问题函数并对漏洞进行简单利用基本过程。

二、实验内容

序	内容	实验情况
1)	PHP 代码审计基础	PHP 代码审计基础漏洞实验 1
2)		PHP 代码审计基础漏洞实验 2
3)		代码审计函数介绍实验
4)	代码审计实践	逻辑错误 metinfo 密码重置实验
5)		sql 注入漏洞禅知 cms 漏洞注入实验

三、实验过程截图（30 分）

注：将下列截图保留，并用简短的话描述实验所得的结果。

1. PHP 代码审计基础漏洞实验 1

步骤一：

demo1.php 展示了 eval 函数的使用：eval 函数将 POST 得到的变量当作 php 代码使用，即调用 system 函数，显示出当前机器的网络信息，如图 1 所示。

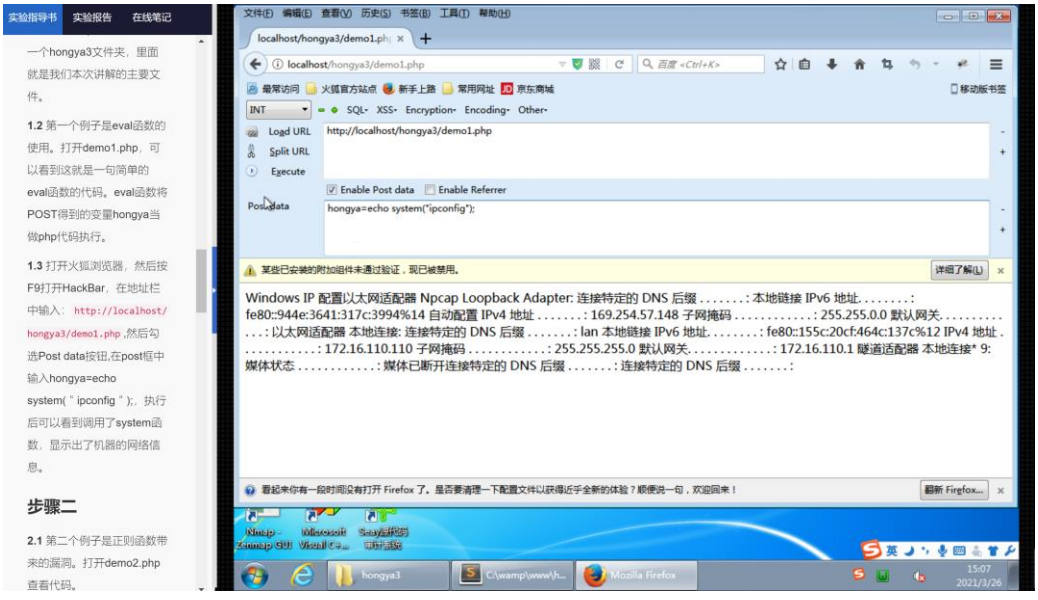


图 1.eval 函数实例

步骤二：

demo2.php 展示了 preg_replace 函数的使用：preg_replace 函数执行一个正则表达式的搜索和替换，由于这里第一个参数添加了\e 修饰符，此时 preg_replace 会将第二个参数当作 php 代码执行。执行显示出 phpinfo()的界面，如图 2 所示。

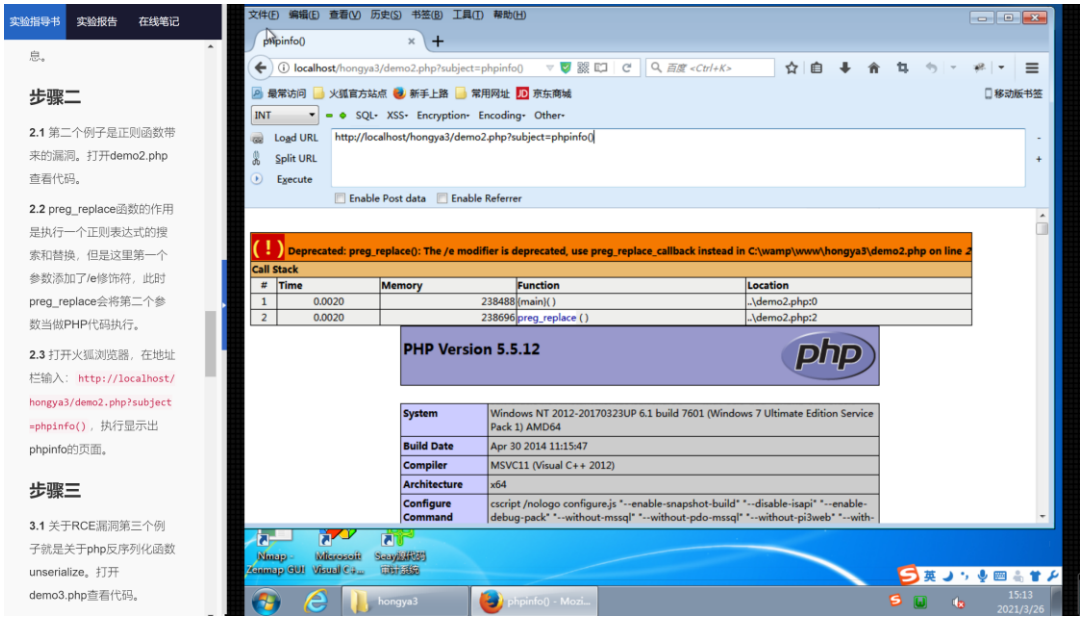


图 2.preg_replace 函数实例

步骤三：

demo3.php 展示了反序列化函数 unserialize 和 RCE 漏洞：执行代码后，左上角的 123，是创建对象 abc 并调用构造函数时 echo '123'。而 phpinfo 页面则是由于对$V 进行反序列化时，php 主动调用了__wakeup 函数，就将 phpinfo 显示到页面了，如图 3 所示。这说明该代码存在远程命令执行漏洞。

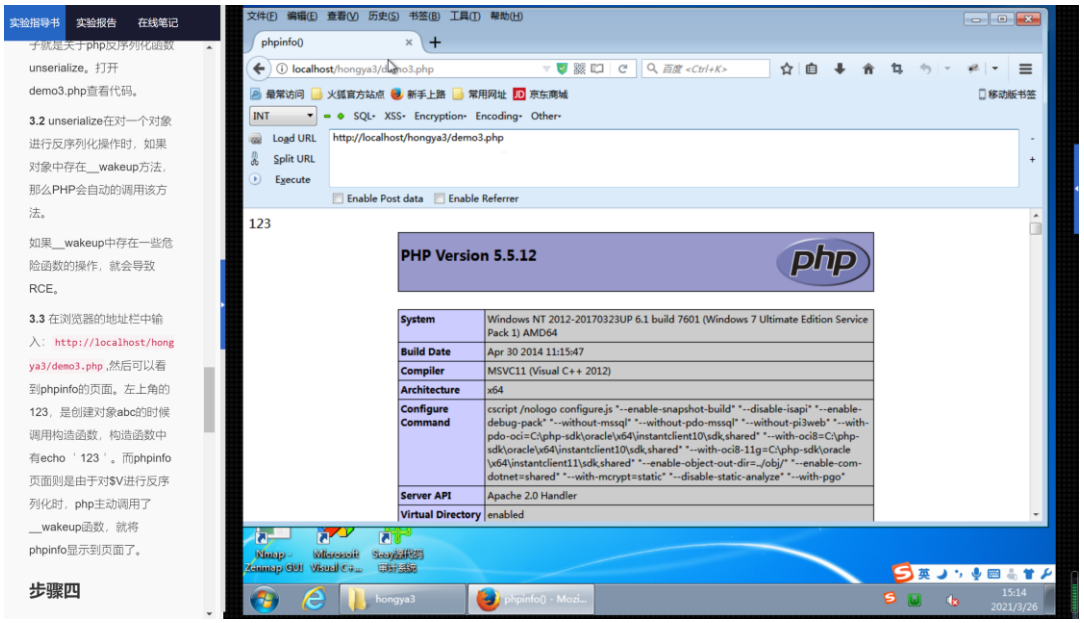


图 3.RCE 漏洞实例

步骤四：

代码里过滤文件格式只能是 gif、jpeg、pjpeg 且文件大小小于 20000 字节。采用 burp suite 进行抓包和修改（首先设置代理地址为 127.0.0.1，端口为 8080），暂时关闭拦截，并选择提交 demo1.php 文件，由于不满足过滤要求所以提示“Invalid file”，如图 4 所示。再打开拦截，此时提交时会发现浏览器没有反应，因为此时 burp suite 拦截了这一 HTTP 请求。

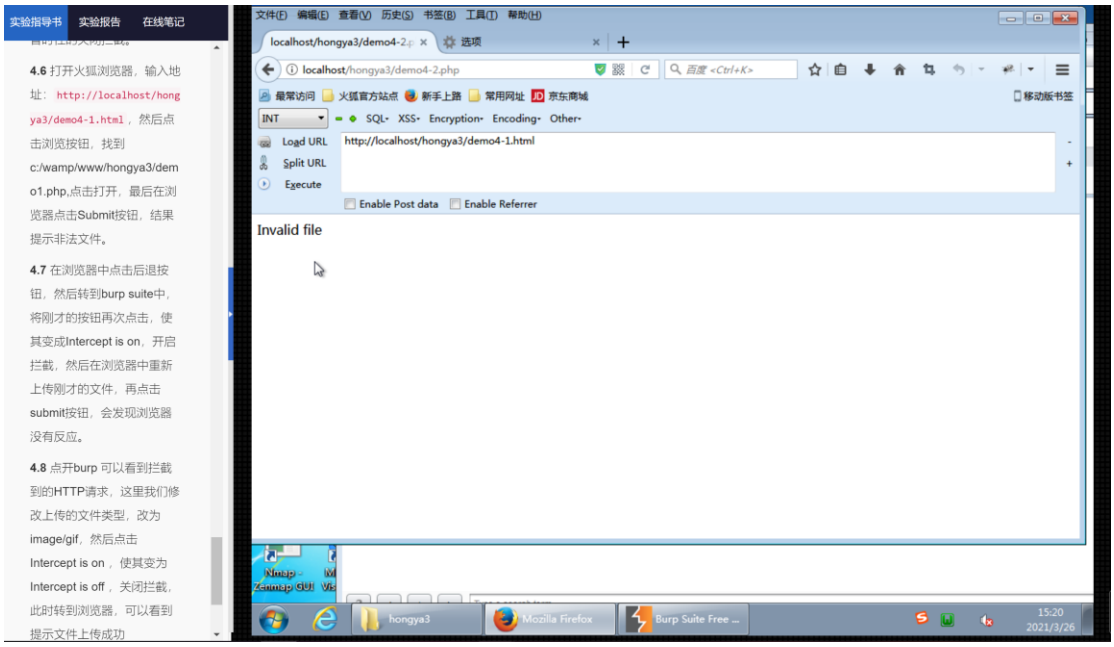


图 4.php 文件提交失败

若我们修改上传的文件类型（Content-Type）为 image/gif（如图 5 所示），然后再次关闭拦截，上传 php 文件发现成功上传，且在 upload 目录下确实看到了 php 文件，如图 6 所示。

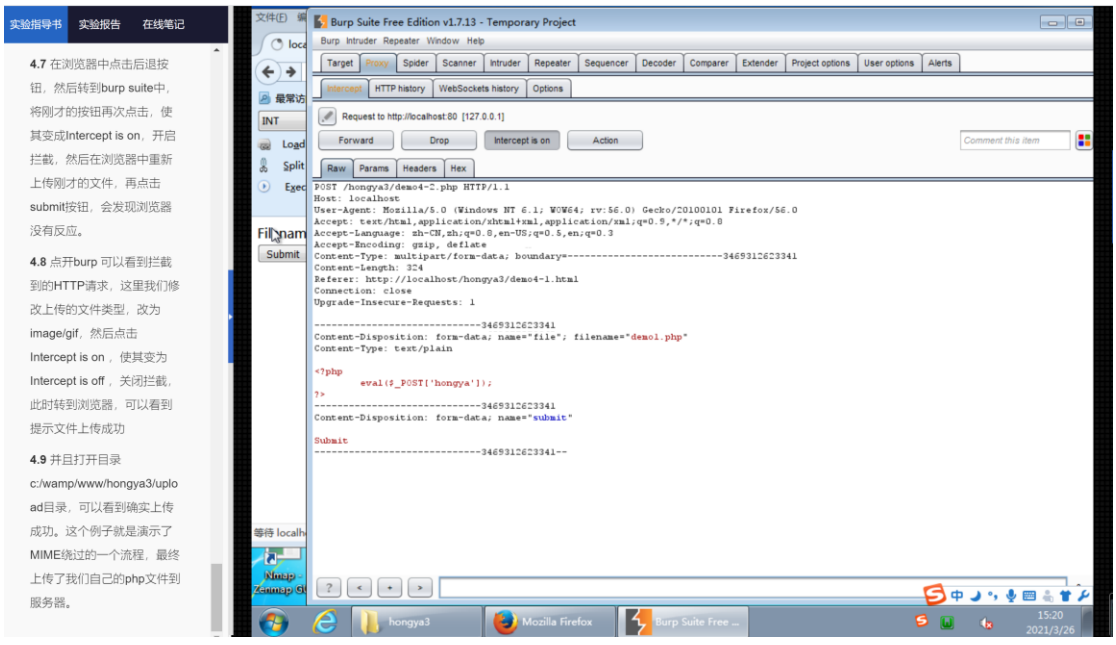


图 5.修改上传文件类型

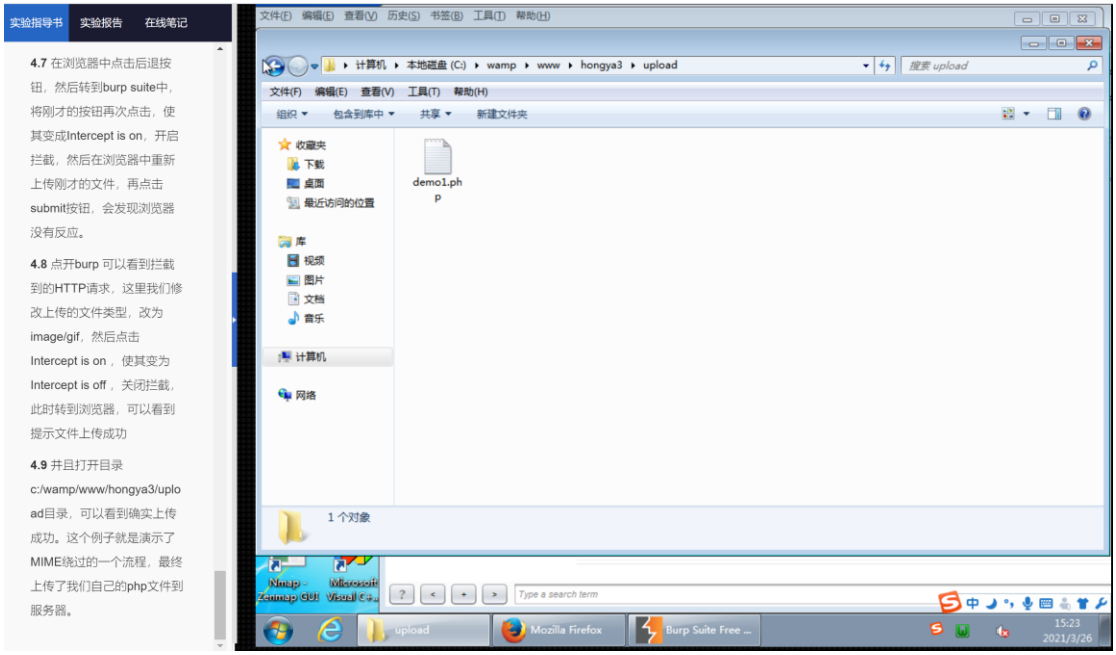


图 6.php 文件成功上传至 upload 目录

2. PHP 代码审计基础漏洞实验 2

步骤一：

demo1.php 的程序逻辑是判断 file 参数不为空则包含变量表示的文件，否则包含默认文件。若给定参数为/etc/passwd，则可以在程序执行后得到密码信息，这就是造成任意文件读取的文件包含漏洞，如图 7 所示。

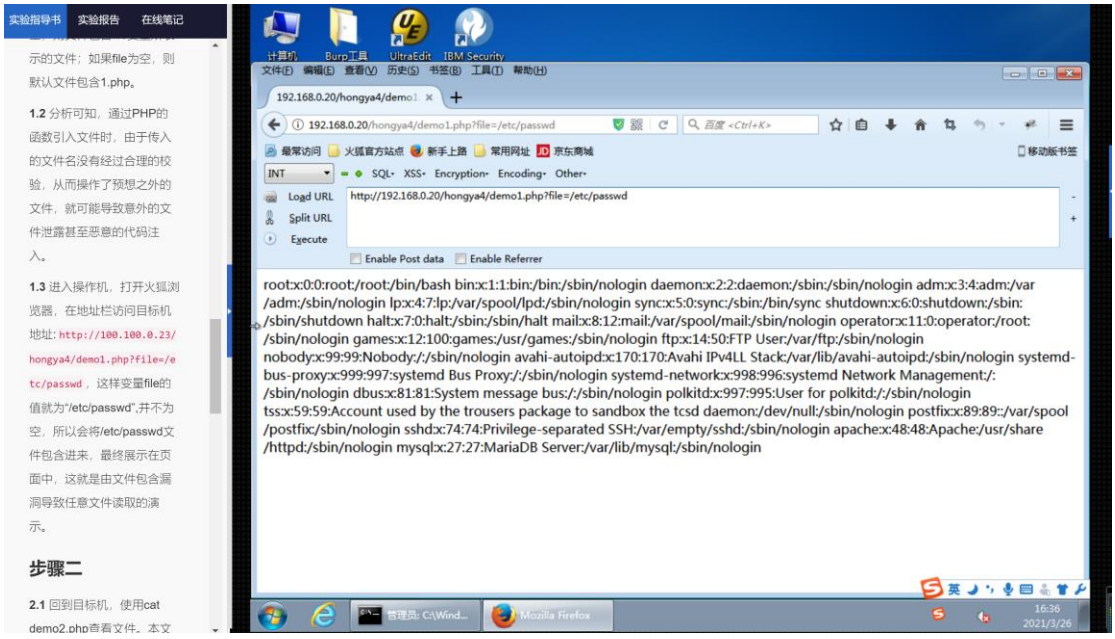


图 7.文件包含漏洞实例 1

步骤二：

demo2.php 的程序逻辑是包含 file 变量表示的文件，若在 Post data 中输入

<?php phpinfo();?>，则执行后直接得到了 phpinfo 页面，如图 8 所示。

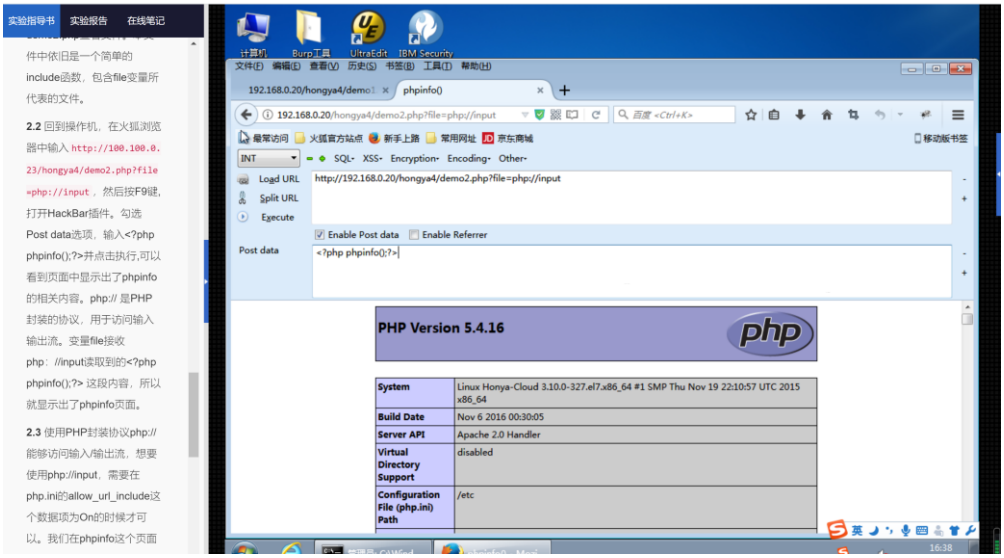


图 8.文件包含漏洞实例 2

步骤三：

demo3.php 的程序逻辑是接收 GET 传入的 name 变量值然后输出。若向 name 传入的值为 user<script>alert(11111)</script>，则执行结果除了输出 user 外，还执行了跟在后面的脚本代码，即弹窗输出 11111，如图 9 所示。这是对输入过滤不足使得未知 html 代码可以任意执行的跨站脚本攻击。

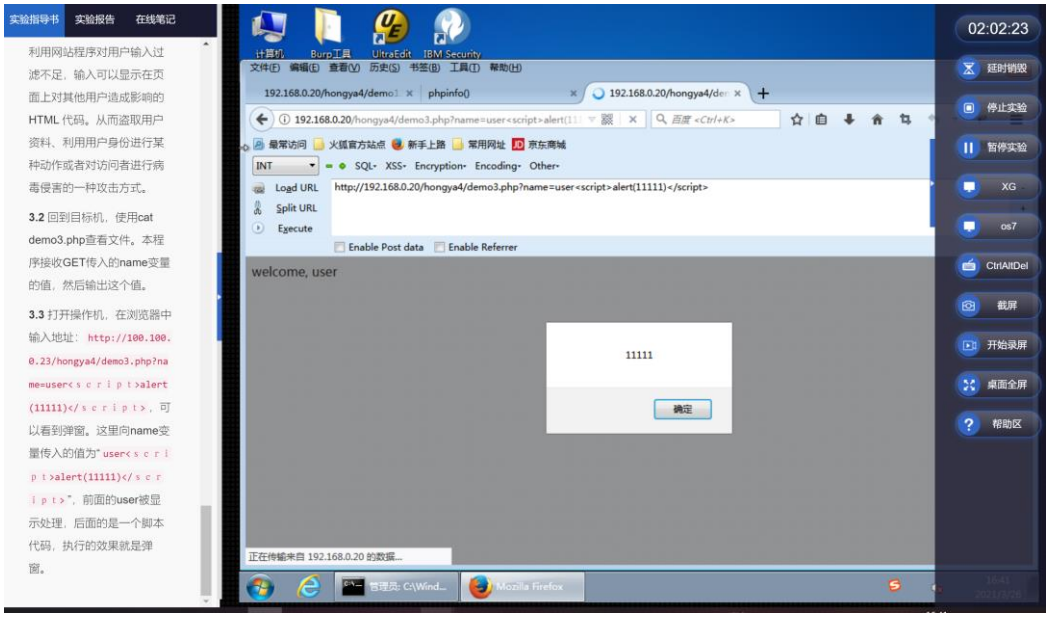


图 9.跨站脚本攻击实例

3. 代码审计函数介绍实验

步骤一：

eval 函数将 POST 的变量当作代码执行，此时若提交 echo system(“whoami”)则执行后会输出当前用户名，暴露个人信息，如图 10 所示。

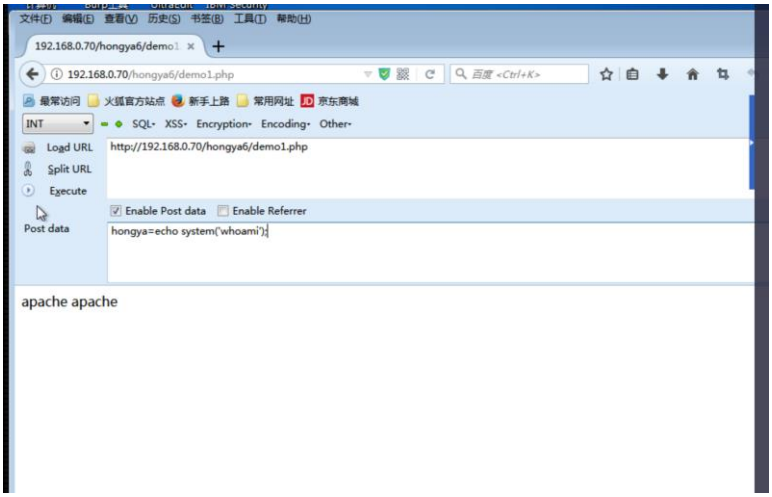


图 10.eval 函数实例

步骤二：

include 函数包含并运行指定文件，缺少对输入的变量的过滤，此时若令变量为/etc/passwd，则执行后该文件将被读取出来并输出，如图 11 所示。

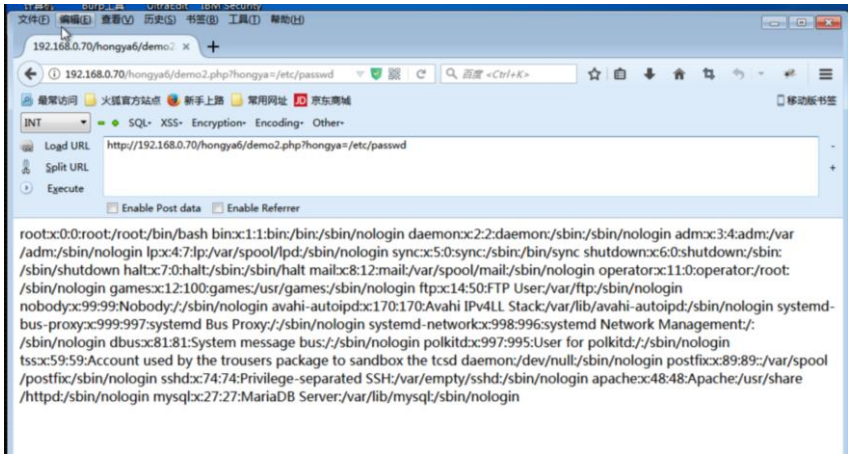


图 11.include 函数实例

步骤三：

file_get_contents 函数将整个文件读入一个字符串中，提交变量同步骤二，则/etc/passwd 文件被读取到字符串中，暴露用户信息，如图 12 所示。

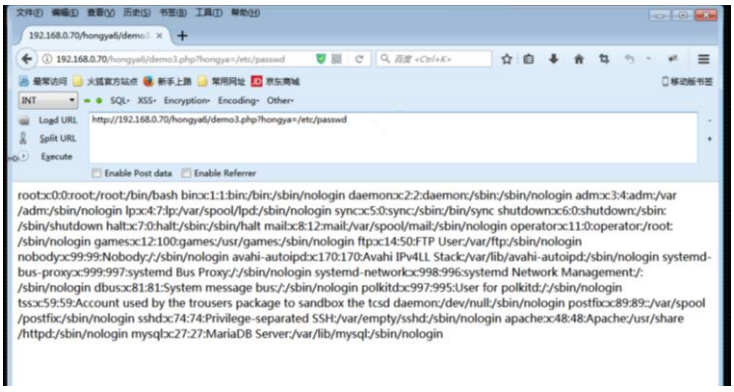


图 12. file_get_contents 函数实例

步骤四：

file_put_contents 函数将一个字符串写入到文件中，如果该文件不存在，默认创建文件并写入字符串。输入字符串为 test，文件参数为 hongya.php，执行后打开 hongya.php 看到 test 已经写入，如图 13 所示。

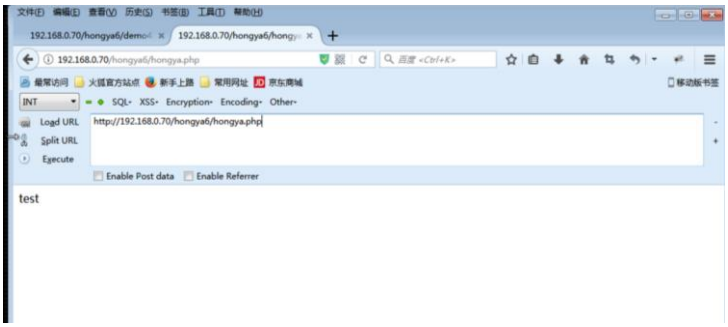


图 13. file_put_contents 函数实例

步骤五：

assert 函数在第一个参数为字符串的时候，会将其当作 PHP 代码来执行。输入变量为 phpinfo()，则 assert 函数将文件内的内容作为 PHP 代码执行，读出 phpinfo 页面并显示，如图 14 所示。

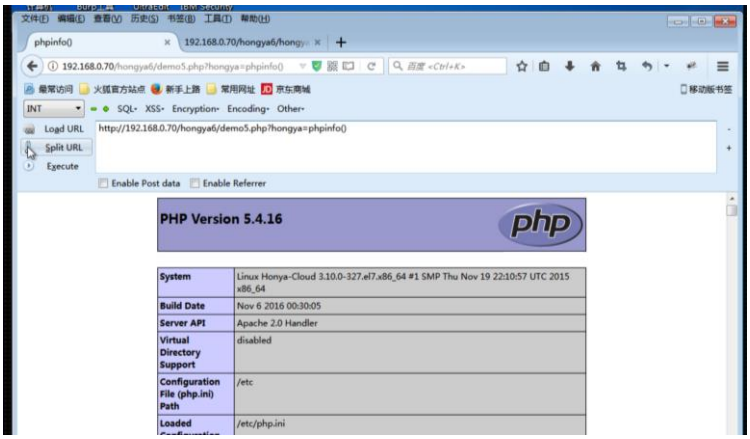


图 14. assert 函数实例

步骤六：

unserialiize 函数对一个变量反序列为 php 值，当变量是一个对象且成功构建对象之后，PHP 会尝试调用__wakeup()成员函数。若变量为 class A 序列化结果，则反序列化后_wakeup()函数会被调用，即执行后输出 hello，如图 15 所示。

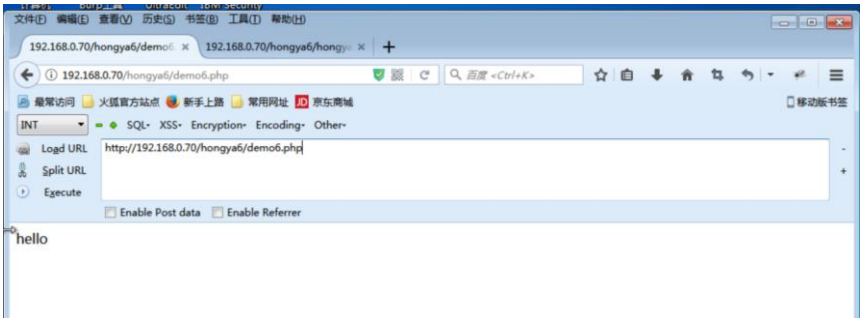


图 15. unserialiize 函数实例

步骤七：

var_dump 函数显示表达式的结构信息，包括表达式的类型与值。执行后可见输出了数组的信息：数组含有三个元素，一个元素是 1，一个元素是 2，第三个元素是一个数组（数组内第一个是 a，第二个是 b，第三个是 c），如图 16 所示。

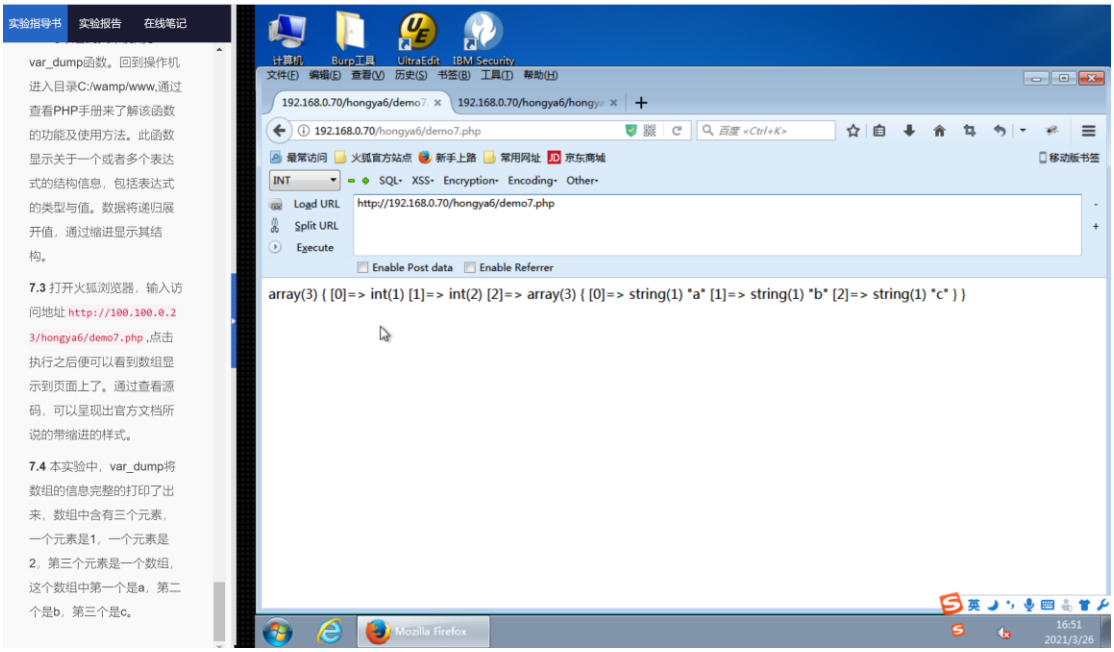


图 16. var_dump 函数实例

4. 逻辑错误 metinfo 密码重置实验

a. 代码审计过程中较为重要的代码段截图

从这里的代码可以看出，$_GET[langset]是可以控制的，构造合适的 SQL 语句能使得程序执行 save_met_cookie()，实现密码重置。

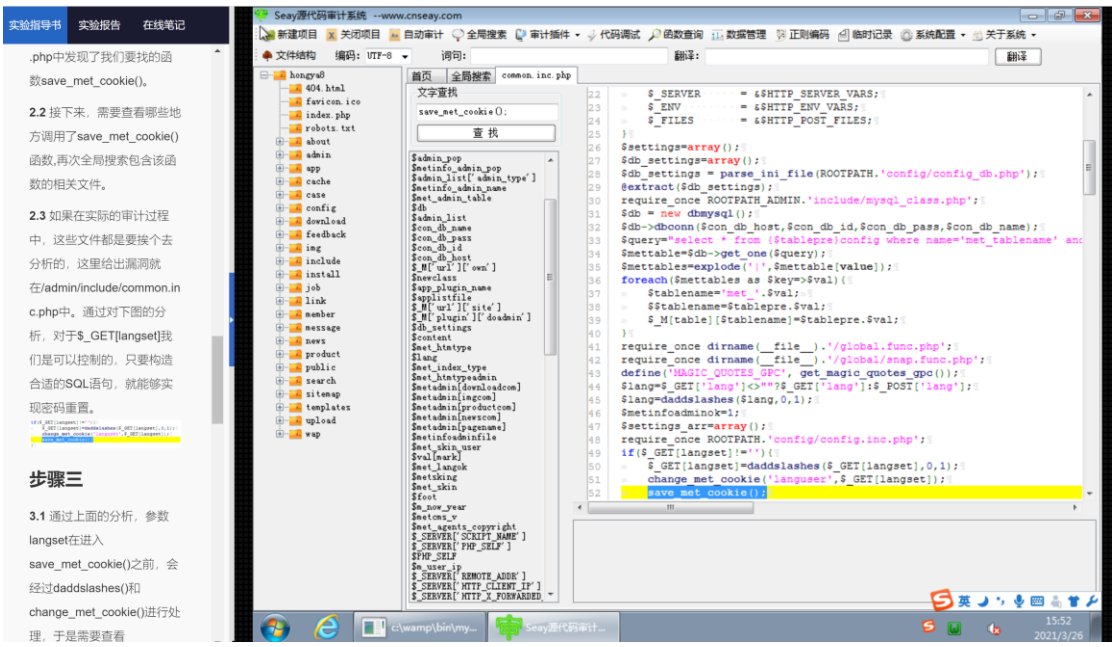


图 17. save_met_cookie()函数实现

而参数 langset 在进入 save_met_cookie()之前，会经过 daddslashes()和 change_met_cookie()处理，而后者并无需要注意的操作，只需检查前者的函数实现：daddslashes()函数中会对输入 string 中的关键词进行过滤，但是对于输入数组，只会对 val 值进行处理，不会对 key 值处理，如图 18 所示。、

从而得出结论：只要将输入封装到 array 中，就能避开过滤并执行 SQL 语句。

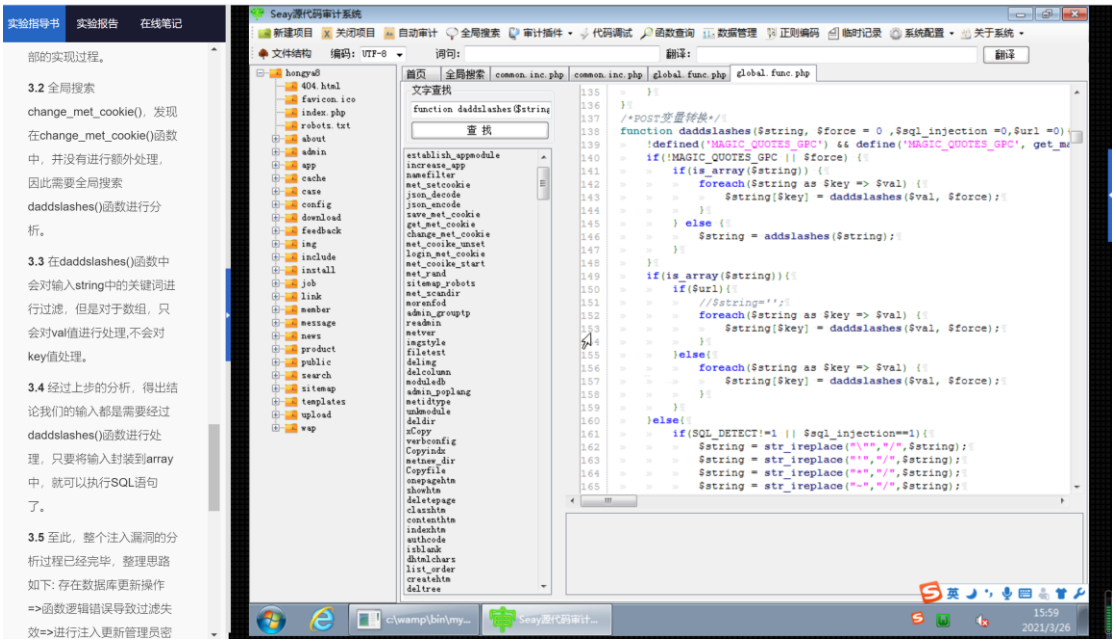


图 18. daddslashes()函数实现

b. 在代码中加入 var_dump 后重置代码时输出的相关信息的浏览器截图

在程序中加入 var_dump 等代码段后，构造出 PAYLOAD，并在浏览器中输入该地址，执行结果如图 19 所示。

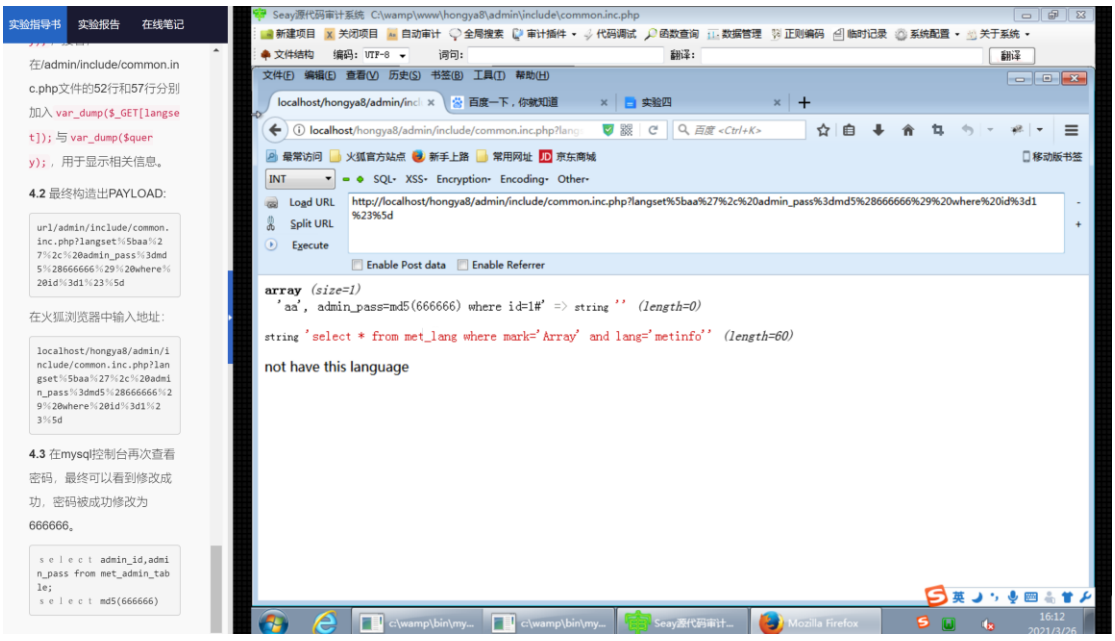


图 19.浏览器截图

c. MySQL 控制台中原始密码与重置密码的 md5 值截图

在 MySQL 控制台中查询密码，并与原始密码做对比，可以看到密码被修改，且新的密码通过比对 md5 值可以确定是 666666，如图 20 所示。

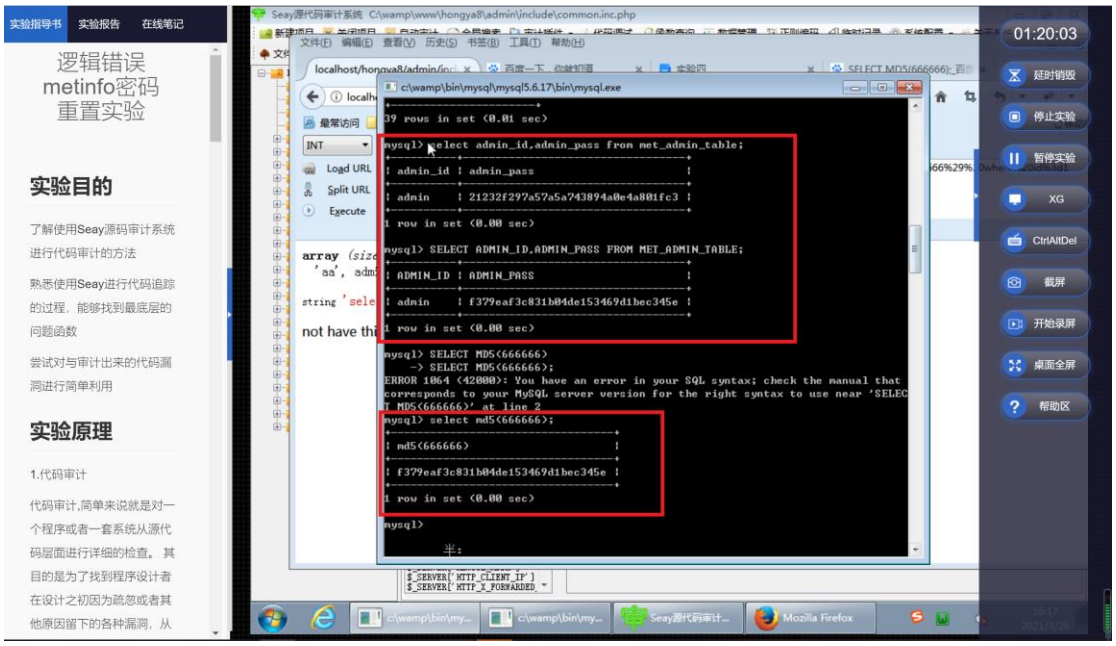


图 20. 密码与 md5 值的截图

四、分析和思考（60 分）

1. 对于防范 PHP 系统命令执行漏洞有哪些建议？（15 分）

系统命令执行漏洞是指：应用程序的某些功能需要调用可以执行系统命令的函数，如果服务器没有充分过滤这些函数或者函数的参数，攻击者就有可能通过命令连接符将恶意命令拼接到正常的函数中，从而随意执行系统命令。

产生原因主要是：服务器没有对执行的命令进行过滤，特别是对一些函数的参数过滤不足，用户可以通过上传变量、代码来随意执行系统命令。

防范建议：

1、尽量使用引号包括参数值，并在拼接前调用 addslashes 进行转义。

2、减少执行外部的应用程序或命令，尽量使用系统内部的自定义函数或函数库来实现相关功能。

3、做严格的参数内容检查，在执行 system 等命令执行前确认参数内容。

4、在 PHP 下禁用高危系统函数，在 php.ini 配置文件中，查找到 disable_functions 并添加需要禁用的函数名。

5、使用 escapeshellarg 函数处理函数参数，将参数中的字符转义，将参数内容限制在一对引号里面，使其无法对当前执行进行截断，实现防范命令注入攻击的目的。

2. 实验中文件类型绕过方法利用的是实验网站哪部分逻辑漏洞，如何检查文件上传会更为有效？ （15 分）

在本实验中，php 程序对上传文件格式进行了过滤，但是过滤参数只检查了 HTTP Header 中的信息，即 **Content-Type** 中的类型，并非文件的真实后缀。若想实现文件类型绕过只需要**修改网络请求中 Content-Type 信息**为 image/gif，即可避开检查，上传任意类型的文件。

检查文件上传的有效方案：

1、使用白名单的方式限定上传文件的后缀，结合使用 MIME Type、后缀检查等技术判断文件类型。

2、对上传的文件目录权限设置为不可执行，即使上传了脚本文件，也不会有执行的权限。

3、限制服务器对上传文件/文件夹的解析。

4、采用更严格的文件类型检查方式来限制文件类型。

5、使用随机数改写文件名和文件路径，防止用户恶意访问并执行预料之外的代码。

3. XSS 跨站脚本有哪些类型？本次实验中是哪种类型？请对其它跨站脚本进行简单举例。（15 分）

XSS 跨站脚本有三种类型：**反射型 XSS、存储型 XSS、DOM 型 XSS**。反射型 XSS 的攻击不经过数据库，存储型 XSS 攻击经过后端并涉及数据库，DOM 型 XSS 攻击不经过后端，通过 URL 传入参数来触发控制。

本次实验中后端没有对 name 变量进行过滤检查，在输入 name=user<script>alert(11111)</script>时执行效果为弹窗显示 11111，属于 **反射型 XSS**。

其他跨站脚本举例：

1、存储型 XSS：

攻击者能够把攻击信息存入服务器的数据库中，造成持久化的攻击。

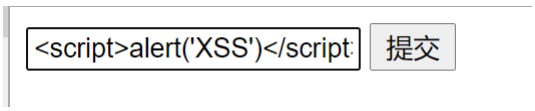
考虑如下代码：

```
<form action="" method="post">
      <input type="text" name="xss"/>
      <input type="submit" value="提交"/>
</form>

<?php
$xss=@$_POST['xss'];
mysql_connect("localhost","root","123");
mysql_select_db("xss");

if($xss!==null){
      $sql="insert into temp(id,payload) values('1','$xss')";
      $result=mysql_query($sql);
      echo $result;
}

?>
```



在表单中输入数据：<script>alert('XSS')</script>，则该内容不直接显示在页面，而是插入到了数据库中，若另外查询数据库中的值时，则会将之前插入的内容显示在页面上。这一攻击方式比反射型危害更大，攻击更持久。

2、DOM 型 XSS：

DOM 型 XSS 从效果上来说也是反射型 XSS，其不同之处在于：DOM 型 XSS 一般和服务器的解析响应没有直接关系，而是在 JavaScript 脚本动态执行的过程中产生的。

如图代码所示：

```
<html>
<head>
<title>DOM Based XSS Demo</title>
<script>
function xsstest()
{
    var str = document.getElementById("input").value;
    document.getElementById("output").innerHTML = "<img src='"+str+"'></img>";
}
</script>
</head>
<body>
<div id="output"></div>
<input type="text" id="input" size=50 value="" />
<input type="button" value="submit" onclick="xsstest()" />
</body>
</html>
```

若此时输入 **x' onerror='javascript:alert(/xss/)** ，并点击按钮即可触发。

执行效果：**submit 按钮的 onclick 事件调用了 xsstest 函数，而在这一函数中修改了页面的 DOM 节点，通过 innerHTML 把一段用户数据当作 HTML 写入到页面中，**造成了 DOM 型 XSS。

4. 分析下列代码有什么漏洞，并举例注入的方法。（15 分）

```php
<?php
$a = addslashes($_GET['a']);
$b = addslashes($_GET['b']);
echo "$a<br/>$b<br/>";
$c = str_replace($a,'',$b);
echo trim($c);
?>
```

首先分析代码结构，$_GET 得到 HTTP 里 GET 到的变量，addslashes 函数的作用是返回转义后的字符，这一函数可以实现对一般 SQL 注入的过滤，因为引进转义可以避免变量中出现单、双引号带来的注入风险。

对于函数 str_replace($search,$replace,$subject)，功能是将 subject 中全部的 search 都被 replace 替换之后并返回最终结果。

综合考虑这个代码段，漏洞来自于这个 str_replace 函数，这一函数导致 SQL 注入漏洞的产生。考虑原本输入 %00’ 时，经过 addslashes 函数过滤后变为 \0\’，字符串注入将被过滤。但是经过 str_replace 匹配 0 并替换为空后，字符串变为 \\’ ，这个字符串表示的是一个转义后的 \ 和一个 ’，可以看出此时单引号成功逃逸，避开了过滤，存在 SQL 注入的风险。

举例注入的方法：

考虑输入数据：b=1%00%27%20and%201=1--+ && a=0

且 SQL 语句为：select * from user1 where id=’

经过 addslashes 函数过滤后数据为：1\0\’ and 1=1--

此时由于加入了转义字符，故不存在 SQL 注入风险

再经过 str_replace 函数，相当于删除了字符串中的 0，则此时数据为：

1\\’ and 1=1--

注意到，此时 SQL 语句实际为：

select * from user1 where id=’1\\’ and 1=1--

返回值为真

若输入数据仅将 1=1 改为 1=2，重复上述操作，则 SQL 语句实际为：

select * from user1 where id=’1\\’ and 1=2--

返回值为假

对比以上两种情况可看出明显的 **SQL 注入漏洞**。

五、实验总结（收获和心得）（5 分）

在本次实验中，我了解并学习了代码审计的思想和具体实现过程，对于 PHP 代码中常见的漏洞有了初步的认识。通过几个实验，我接触到了 RCE 漏洞、任意文件上传漏洞、文件包含漏洞、SQL 注入漏洞、XSS 跨站脚本和 PHP 中许多函数引起的安全漏洞。在过去的学习过程中，我对于 web 安全只有粗浅的概念，在研究具体的漏洞实例和运行了相关 PHP 代码之后，我深刻体会到漏洞对安全的危害性，并且认识到在编写代码、日常安全维护的过程中要时时刻刻保持清醒的头脑，仔细排查安全风险，思考可能产生漏洞的疏漏之处。

我也通过实验熟悉了代码审计，审计的目的是为了找到代码设计时可能留下的漏洞、隐患，从源代码层面对一个程序进行详细的检查，排除安全风险，最终提供代码修订措施和建议。从实验四我们可以看出，代码审计不是简单的检查一个程序的代码，对于一个系统而言，漏洞的产生可能是多个文件、函数互相调用的结果，所以审计的过程需要综合考虑这些代码，并分析攻击者可能选择的路径，探索可能存在的风险。

这次实验加深了我对代码审计和漏洞挖掘的兴趣，希望我能在这次实验学习到的知识基础上继续深入学习，探索研究。

六、尚存问题或疑问、建议（5 分）

1、在实验中，DHCP 池分配 IP 地址是随机的，但在完成目标机和操作机的实验时要求两个机器在同一个网段，这在进行实验过程中给我们造成了很大的困扰。为了确保实验的正常进行，我们必要在实验前反复重启实验平台，打开虚拟机检查二者的 IP 地址是否在同一个网段。而我因为运气不好，花了半个小时，重启随机了很多次才遇上了 IP 地址符合要求的情况……希望实验平台之后能优化 DHCP 协议和具体的 IP 地址分配。

2、在实验二和实验三，操作机和目标机进行交互时，注意 IP 地址需要按照实际分配的地址填写，不能照抄指导手册里面的数字。（具体 IP 地址查询，Windows 系统用 ipconfig 命令，Linux 系统用 ifconfig 命令）

3、在实验中测试代码输入时，如果需要输入大量实验要求的文本，由于实验平台的虚拟机不具备剪贴板功能，所以不得不一个个字符手敲进文本框，建议能在实验平台内开启剪贴板功能，方便同学们在输入长文本时，复制相关指令。