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

BUG_Author: @wfgo

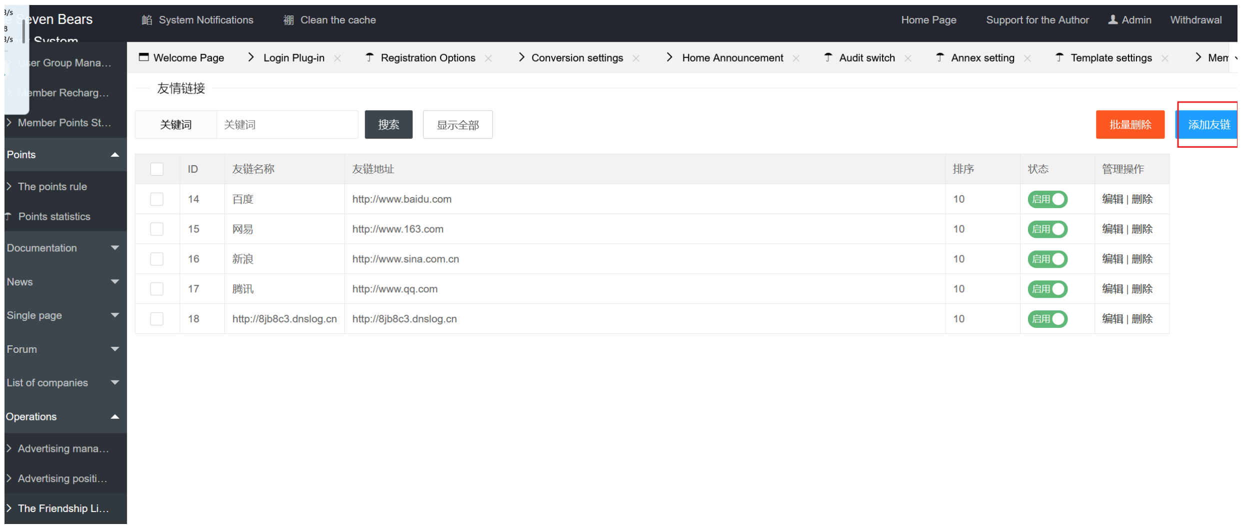
Affected Version: Seven Bears Library CMS <2023

Vendor: https://gitee.com/mirweiye

Vulnerability Files:

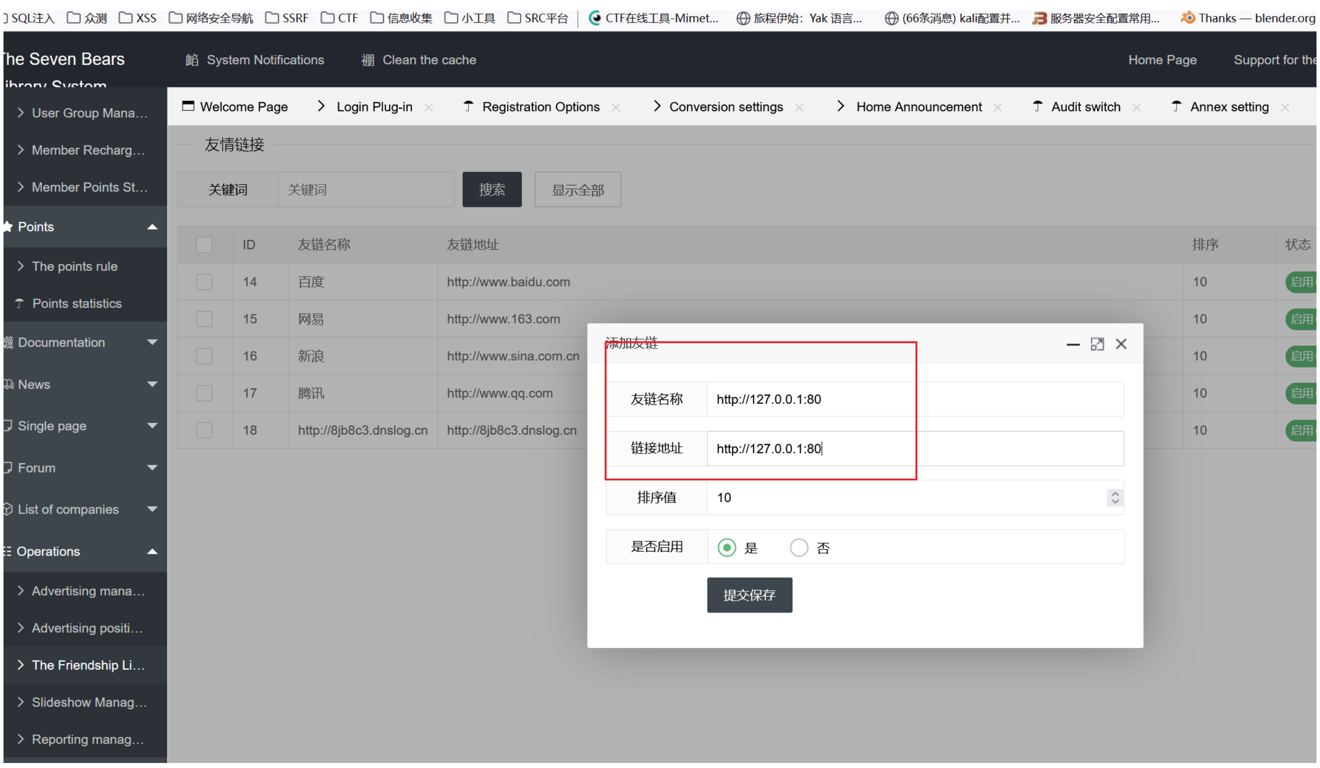
- wenkucms\app\Lib\Action\home\docAction.class.php
- wenkucms\app\Lib\ORG\Http.class.php
- wenkucms\wkcms\Extend\Engine\Sae\Common\sae_functions.php
- wenkucms\wkcms\Extend\Engine\Sae\DefaultApp\Lib\Action\IndexAction.class.php

Description:

1. In the background interface, click Add Ally in the Add Link function point, as shown in the figure
2. 

	ID	友链名称	友链地址	排序	状态	管理操作
	14	百度	http://www.baidu.com	10	启用	编辑 | 删除
	15	网易	http://www.163.com	10	启用	编辑 | 删除
	16	新浪	http://www.sina.com.cn	10	启用	编辑 | 删除
	17	腾讯	http://www.qq.com	10	启用	编辑 | 删除
	18	http://8jb8c3.dnslog.cn	http://8jb8c3.dnslog.cn	10	启用	编辑 | 删除

At this time, if you add a private IP address, as shown in the figure

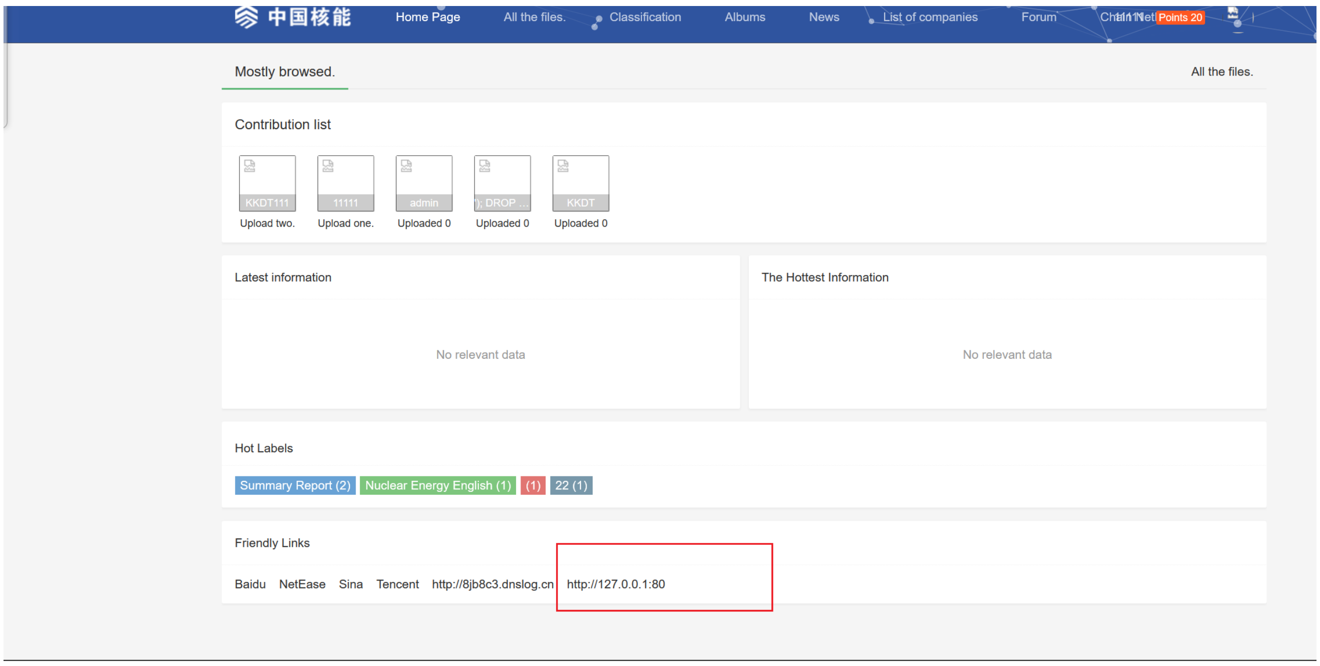


	ID	友链名称	友链地址	排序	状态
	14	百度	http://www.baidu.com	10	启用
	15	网易	http://www.163.com	10	启用
	16	新浪	http://www.sina.com.cn	10	启用
	17	腾讯	http://www.qq.com	10	启用
	18	http://8jb8c3.dnslog.cn	http://8jb8c3.dnslog.cn	10	启用

添加友链

友链名称	http://127.0.0.1:80
链接地址	http://127.0.0.1:80
排序值	10
是否启用	[✓ 是] 否

After that, you can jump to another interface on the intranet, as shown in the figure



Friendly Links

Baidu NetEase Sina Tencent http://8jb8c3.dnslog.cn http://127.0.0.1:80

代码分析

1. ## 1. File 'doc_cateAction.php'

   1. Code Function Overview: This file defines the 'doc__cateAction' class, where the 'ajax__getsubcate' method is used to obtain the subcategory information of the classified ('status = 1') under the classification from the database based on the passed classification ID ('$id') and return the operation results and sub-classification data in JSON format.

   2. Potential vulnerability analysis

   :

   - There is no direct server-side attempt to initiate an external request in this file, so there is no typical SSRF vulnerability trigger point at this time.
   - However, if user input (e.g., '$id') is used to construct server-side requests (e.g., '$id' to concatenate request URLs when calling external APIs) in this class or related functionality, and the input is not rigorously validated and filtered, an attacker could create an SSRF vulnerability by constructing a malicious '$id' value to make the server make a request to any destination address.

2.# file contains a file of the 'sae_storage_root' function

1. Function Overview: The sae__storage__root function is used to obtain the domain name address stored by SAE (Sina App Engine). If 'SAE__CACHE__BUILDER' is defined, a string of a specific format is returned; Otherwise, instantiate the 'SaeStorage' class and get the URL address of the storage.

2. Potential vulnerability analysis

:

- The function itself does not directly receive user-controllable input to construct the request address, and there is no direct SSRF risk.
- However, if the return value of the function is used in subsequent server-side request construction, and the returned URL is not validated and filtered during use, if an attacker can tamper with the configuration or related data stored by SAE in some way to make the 'sae__storage__root' function return a malicious URL address, then the server may make a malicious request, thus triggering an SSRF vulnerability.

3.### 3. File 'IndexAction.php'

1. **Code Function Overview**: This file defines the 'IndexAction' class, which contains several methods for testing various functions of the SAE service simulator, such as database operations ('mysql' method), caching tests ('scache', 'fcache' methods), file upload and download ('upload', 'storage' methods), task queues ('tq' methods), mail sending ('mail' methods), URL scraping (' fetchurl' method) and so on.

2. Potential vulnerability analysis

:

- 'Fetchurl' method

:

- This method uses the 'fetch' method of the 'SaeFetchurl' class to scrape the content of the specified URL ('http://sina.cn') and output it. The URL here is hard-coded, but if the URL is subsequently changed to user input control without strict validation and filtering of the input, an attacker can enter an arbitrary URL and cause the server to make a request to the destination address they specify, triggering an SSRF attack. For example, an attacker can enter a private IP address or a malicious server address to obtain intranet information or perform other malicious operations.

- 'tq' method (task queue)

:

- In the 'tq' method, when adding a task using the 'SaeTaskQueue' class, pass in the task's URL via the '$t->addTask' method. At present, the incoming URL is based on a relatively fixed address concatenated by '$_SERVER['HTTP_HOST']' and '**URL**', but if the content of these parts can be controlled by the user in practical application, or if there is a vulnerability in the concatenation logic, the attacker may construct a malicious task URL to let the server execute malicious requests, which has potential SSRF risks.

- **Other methods**: Although other methods such as 'storage' (file storage operation), 'mail' (mail sending), etc., do not have obvious SSRF risks, if the relevant parameters (such as the recipient address of the email sent, the remote address of the file storage, etc.) can be controlled by the user input in the subsequent function expansion, and no security processing is carried out, SSRF vulnerabilities may also be introduced.

1. ### (2) Attack vector (potential)
    1. Control request URLs: If the request URLs involved in the above potential vulnerabilities can be controlled by an attacker, for example, in the 'fetchurl' method, the attacker can construct a URL to an intranet server, so that the server can access internal resources and obtain sensitive information, such as internal databases, file systems, etc.
    2. Tampering with configuration or data: In the case of relying on specific configuration or data (such as the SAE storage configuration that the 'sae_storage_root' function depends on) to generate the request address, if an attacker can tamper with these configurations or data so that the generated request address points to a malicious server, the server will send a request to the malicious address, which can be used by the attacker to carry out phishing attacks, obtain server privileges, etc.
    3. Exploit Feature Extension Vulnerability: When extending existing features, if the new feature uses user input to construct server-side requests and does not follow the security development specifications for input validation and filtering, an attacker can exploit the vulnerabilities introduced by these new features to construct malicious requests and implement SSRF attacks.

1. ### (3) Exploit Scenarios (Potential)
    1. Intranet information detection: An attacker constructs a request URL containing an intranet IP address, exploits the SSRF vulnerability to allow the server to send a request to the intranet, and probes information such as surviving hosts, open ports, and services on the intranet to prepare for further attacks. For example, an attacker could try to access a private address such as 'http://192.168.1.100:80' to see if they can get a response.

    2. Sensitive data theft: If there are accessible sensitive data storage systems (such as databases, file servers, etc.) on the internal network, attackers can construct requests for servers to access these systems and obtain sensitive data. For example, if there is an unauthorized access vulnerability in an internal database, an attacker can construct a SQL query request to obtain information such as user accounts and passwords in the database through the server.
    3. Attack on internal systems: Attackers can exploit SSRF vulnerabilities to allow servers to make malicious requests to internal web applications, such as SQL injection, XSS attacks, etc., to tamper with data and obtain system privileges. For example, construct a request URL that contains a malicious SQL statement and have the server make a request to an internal database application to execute the malicious SQL statement.
    4. Port scanning: An attacker can have a server make a request to a series of IP addresses and ports, perform a port scan, find out which ports are open and services are running in the internal network, so as to determine the targets and methods of further attacks.

1. ## III. Impact of the vulnerability
    1. **Information Leakage Risk**: It may lead to the leakage of internal sensitive information, such as user data, system configuration information, business data, etc., bringing serious privacy and security problems to users and enterprises.
    2. System security threats: Attackers can exploit SSRF vulnerabilities to attack internal systems, obtain system permissions, tamper with data, destroy the availability and integrity of systems, and affect the normal operation of services.
    3. Network Security Risks: SSRF attacks can be used to probe and attack internal networks, breaching the security perimeter of the network, exposing the internal network to risks, and potentially threatening the security of the entire network.
    4. Reputational damage: In the event of a security incident caused by an SSRF vulnerability, the reputation of the enterprise can be severely impacted, and the trust of users in the enterprise may decrease, which may lead to business loss and customer churn.

   ## Fourth, the affected environment
    1. Application Version: All application versions that use the above code may be affected by the SSRF vulnerability if they have the potential insecurity factors described above. In particular, the risk of exploiting vulnerabilities is higher if the relevant security issues are not fully considered and addressed after functional expansion or configuration modification.

2. Runtime: This vulnerability has little to do with the runtime environment, and mainly depends on the code implementation and configuration of the application. Whether in a development, test, or production environment, vulnerabilities can be exploited by attackers.
3. Server Configuration: If the server's network configuration allows the server to access the internal network, and the application has an SSRF vulnerability, an attacker can exploit the vulnerability to exploit the internal system. Therefore, the server's network configuration, such as firewall settings, access control policies, etc., can also affect the extent of exploitation.

V. Suggestions for fixing vulnerabilities

1. Input validation and filtering

:

- Rigorous validation and filtering of all user input that may be used to construct server-side requests (e.g., URL input in the 'fetchurl' method, relevant parts of the task URL in the 'tq' method, etc.). You can use a whitelist mechanism to allow access to only specified domain names or IP addresses

```
$allowed_domains = ['sina.cn', 'example.com']; // 定义允许访问的域名白名单
$input_url = $_GET['url']; // 假设从 GET 参数获取 URL
$parsed_url = parse_url($input_url);
if ($parsed_url && in_array($parsed_url['host'], $allowed_domains)) {
    // 允许访问
    $f = new SaeFetchurl();
    echo $f->fetch($input_url);
} else {
    // 拒绝访问
    echo '不允许访问该地址';
}
```

1. **输出编码和转义**：对所有输出到页面或用于进一步处理的用户输入进行编码和转义，防止恶意代码注入和 URL 篡改。例如，在显示用户输入的 URL 时，使用 `htmlspecialchars` 函数进行编码，防止 XSS 攻击和 URL 混淆。

2. **安全的请求库和配置**：在使用请求库（如 `SaeFetchurl`、`SaeTaskQueue` 等）时，确保使用安全的版本，并正确配置相关选项。例如，对于 `SaeFetchurl`，可以增加对请求 URL 的合法性检查，限制协议类型（只允许 `http` 和 `https` 等安全协议）。
3. **加强配置管理**：对于依赖配置文件或特定数据来生成请求地址的情况（如 `sae_storage_root` 函数依赖的 SAE 存储配置），加强对配置文件的访问控制和数据验证。定期检查配置文件的完整性和安全性，防止配置被篡改。
4. **监控和审计**：建立完善的监控和审计机制，记录所有的服务器端请求信息，包括请求 URL、请求参数、响应结果等。及时发现异常的请求行为，并进行分析和处理。可以使用日志记录工具和安全监控系统来实现这一目标。

六、漏洞检测建议

1. 手动测试

   ：

   - 使用浏览器的开发者工具或其他工具（如 Postman），尝试构造恶意的请求 URL，观察服务器的响应。例如，在 `fetchurl` 方法对应的功能页面，尝试输入内网地址或恶意服务器地址，查看服务器是否会发起请求以及返回的结果。
   - 对涉及用户输入的功能进行全面测试，检查是否存在输入验证和过滤不严格的情况。尝试输入特殊字符、恶意脚本等，看是否会影响服务器的正常运行或导致安全问题。
2. **自动化扫描**：使用专业的漏洞扫描工具（如 OWASP ZAP、Nessus 等）对应用程序进行全面扫描。这些工具可以模拟各种攻击场景，检测是否存在 SSRF 漏洞，并提供详细的漏洞报告和修复建议。在扫描过程中，确保工具能够覆盖所有可能存在 SSRF 风险的功能点。
3. 代码审查

   ：对应用程序的代码进行详细的审查，特别是涉及服务器端请求构造的部分。检查是否存在未进行安全处理的代码逻辑，确保输入验证、过滤、输出编码等安全措施得到正确实施。审查代码时，可以关注以下几个方面：

   - 是否有用户输入直接用于构造请求 URL。
   - 是否对请求 URL 进行了合法性检查和过滤。
   - 是否对请求库的使用进行了安全配置。
   - 是否存在依赖配置或数据来生成请求地址的情况，以及对这些配置和数据的安全性处理。

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:; ');