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

BUG_Author: @wfgo

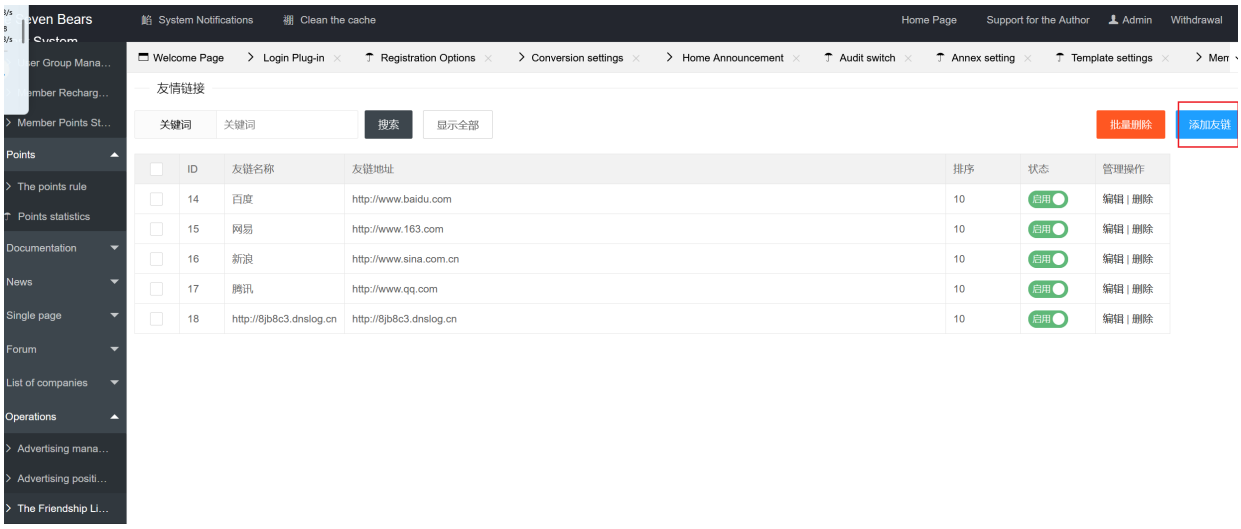
Affected Version: Seven Bears Library CMS <2023

Vendor: https://gitee.com/mirweiye

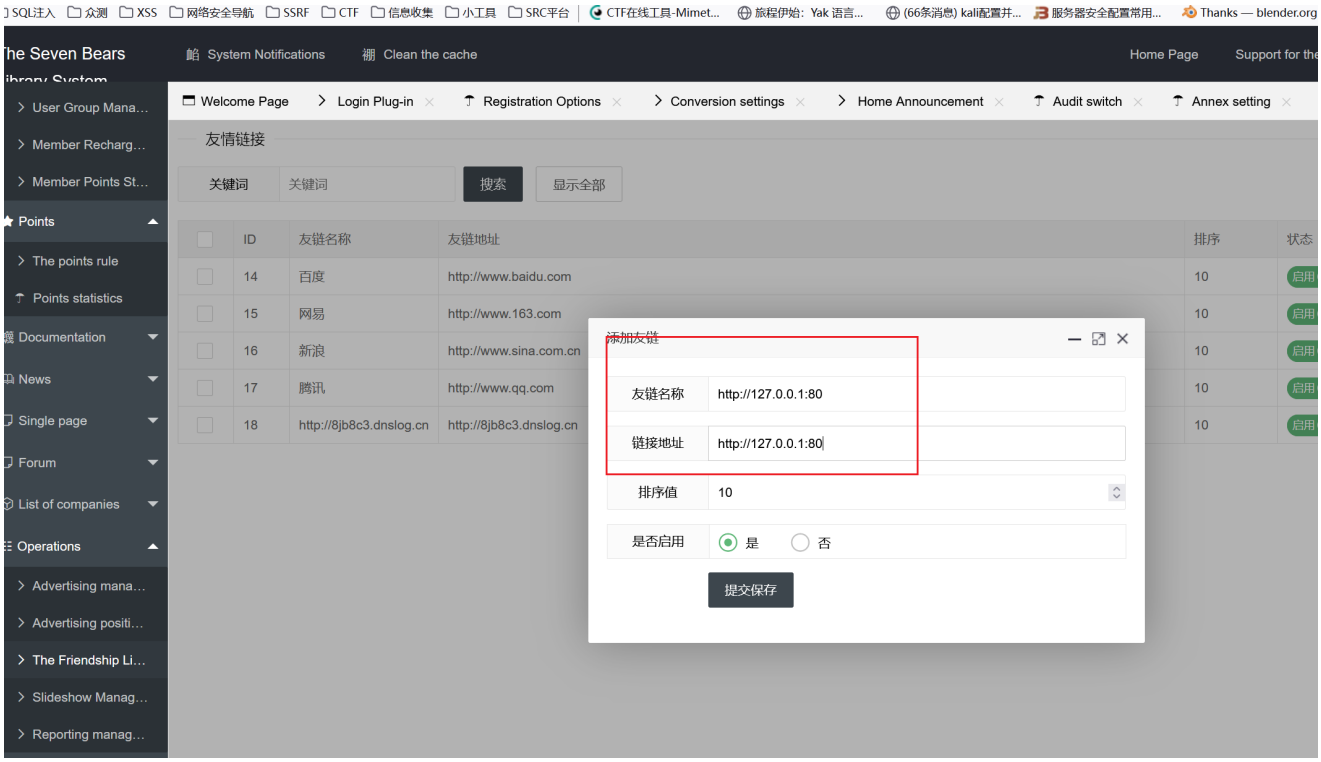
Vulnerability Files:

- 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

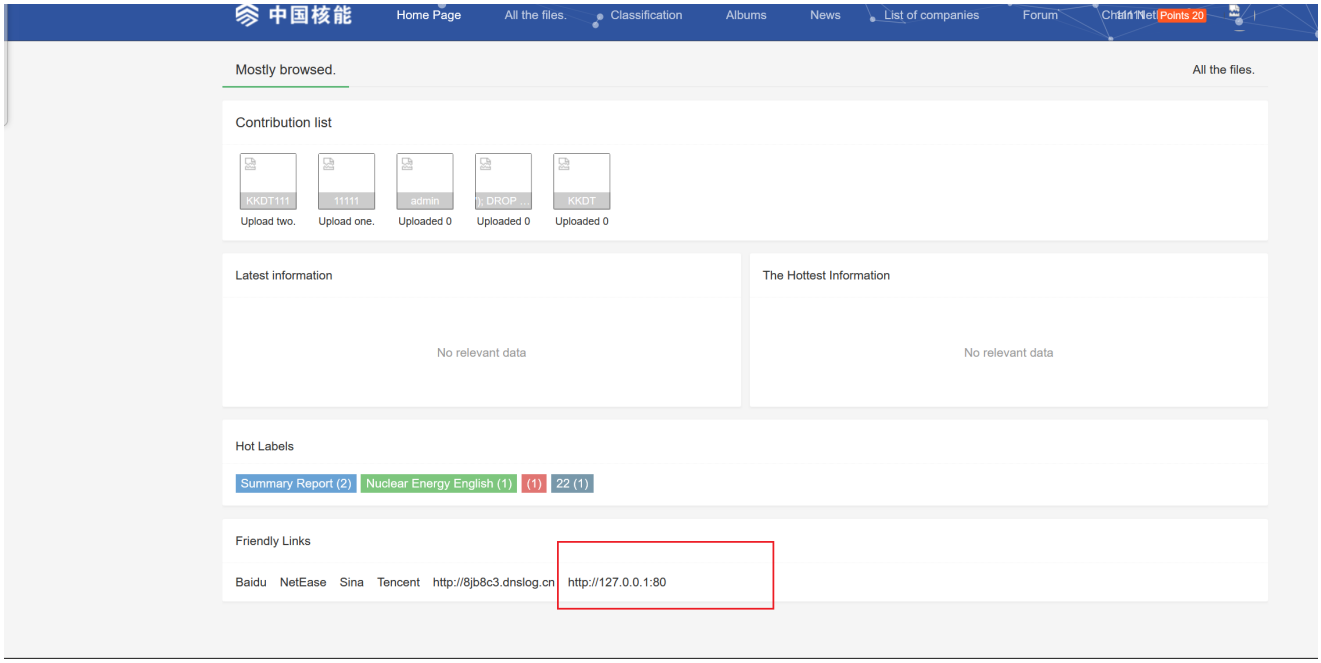
Description:

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

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



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



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

```php
$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. Output Encoding and Escape: Encode and escape all user input that is output to a page or used for further processing, preventing malicious code injection and URL tampering. For example, when displaying a URL entered by a user, use the 'htmlspecialchars' function for encoding to prevent XSS attacks and URL obfuscation.

2. Secure Request Libraries and Configurations: When using request libraries (e.g. 'SaeFetchurl', 'SaeTaskQueue', etc.), make sure you are using a secure version and configure the options correctly. For example, for 'SaeFetchurl', you can add legitimacy checks for the request URL and limit the type of protocol (only secure protocols such as 'http' and 'https' are allowed).
3. Enhanced Configuration Management: Strengthen access control and data validation for profiles for situations that rely on profiles or specific data to generate request addresses, such as SAE storage configurations that the 'sae_storage_root' function relies on. Regularly check the integrity and security of configuration files to prevent configuration tampering.
4. Monitoring and auditing: Establish a complete monitoring and auditing mechanism to record all server-side request information, including request URLs, request parameters, response results, etc. Detect abnormal request behaviors in a timely manner, analyze and handle them. This can be achieved using logging tools and security monitoring systems.

6. Suggestions for vulnerability detection

1. Manual testing

:

- Use your browser's developer tools or other tools, such as Postman, to try to construct a malicious request URL and observe the server's response. For example, on the function page corresponding to the 'fetchurl' method, try to enter a private IP address or a malicious server address to check whether the server initiates a request and the result returned.
- Perform a thorough test of features involving user input to check for lax input validation and filtering. Try entering special characters, malicious scripts, etc., to see if it affects the normal operation of the server or causes security issues.

2. Automated Scanning: Perform a comprehensive scan of your application using professional vulnerability scanning tools such as OWASP ZAP, Nessus, etc. These tools simulate a variety of attack scenarios, detect the presence of SSRF vulnerabilities, and provide detailed vulnerability reports and remediation recommendations. During the scanning process, make sure that the tool is able to cover all the function points that may pose an SSRF risk.

3. Code review

: A detailed review of the application's code, especially those that involve server-side request construction. Check for code logic that has not been securely processed to ensure that security measures such as input validation, filtering, and output encoding are properly implemented. When reviewing your code, you can focus on the following areas:

- Whether there is user input that is directly used to construct the request URL.
-

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:; ');