SSRF in Glances IP Plugin via public_api leads to credential leakage

High **nicolargo** published **GHSA-g5pq-48mj-jvw8** 2 days ago

Package

glances (pip)

Affected versions	Patched versions
Affected versions: 4.5.2 (latest, unpatched)	>= 4.5.4

Description

Summary

A Server-Side Request Forgery (SSRF) vulnerability exists in the Glances IP plugin due to improper validation of the public_api configuration parameter. The value of public_api is used directly in outbound HTTP requests without any scheme restriction or hostname/IP validation.

An attacker who can modify the Glances configuration can force the application to send requests to arbitrary internal or external endpoints. Additionally, when public_username and public_password are set, Glances automatically includes these credentials in the Authorization: Basic header, resulting in credential leakage to attacker-controlled servers.

This vulnerability can be exploited to:

Access internal network services (e.g., 127.0.0.1, 192.168.x.x)
Retrieve sensitive data from cloud metadata endpoints (e.g., 169.254.169.254)
Exfiltrate credentials via outbound HTTP requests

The issue arises because public_api is passed directly to the HTTP client (urlopen_auth) without validation, allowing unrestricted outbound connections and unintended disclosure of sensitive information.

Details

The vulnerability exists in the Glances IP plugin where the public_api configuration value is used to fetch public IP information. This value is read directly from the configuration file and passed to the HTTP client without any validation.

Root Cause
In glances/plugins/ip/**init**.py, the public_api parameter is retrieved from configuration and later used to initialize a background thread responsible for making HTTP requests:

```
self.public_api = self.get_conf_value("public_api", default=[None])[0]

self.public_ip_thread = ThreadPublicIpAddress(
    url=self.public_api,
    username=self.public_username,
    password=self.public_password,
    refresh_interval=self.public_address_refresh_interval,
)
```

There is no validation performed on:

- URL scheme (e.g., http, https, file)
- Hostname or resolved IP address
- Internal or restricted IP ranges
- Unsafe HTTP Request Handling

The request is executed via urlopen_auth() in glances/globals.py:

```
def urlopen_auth(url, username, password, timeout=3):
    return urlopen(
        Request(
            url,
            headers={
                'Authorization': 'Basic ' +
                base64.b64encode(f'{username}:{password}'.encode()).decode()
            },
        ),
        timeout=timeout,
    )
```

This function:

- Accepts any URL passed to it
- Automatically attaches a Basic Authorization header
- Does not enforce any restrictions on destination

PoC

SSRF via public_api (Glances IP Plugin)
Prerequisites
Glances installed
Two terminals
Step 1 Start listener (Terminal 1)
`nc -lvnp 9999`

Step 2 Create malicious config (Terminal 2)
`mkdir -p ~/.config/glances`

`cat > ~/.config/glances/glances.conf << 'EOF' [ip] public_disabled=False`
`public_api=http://127.0.0.1:9999/ssrf-poc public_username=apiuser`
`public_password=S3cr3tP@ss EOF`

Step 3 Start Glances
glances --webserver
Step 4 Observe SSRF request (Terminal 1)
`GET /ssrf-poc HTTP/1.1 Host: 127.0.0.1:9999 User-Agent: Python-urllib/3.x`

`Authorization: Basic YXBpdXNlcjpTM2NyM3RQQHNz`
Step 5 Decode leaked credentials
`echo "YXBpdXNlcjpTM2NyM3RQQHNz" | base64 -d`

Output:
`apiuser:S3cr3tP@ss`
Step 6 Confirm data via API
`curl -s http://127.0.0.1:61208/api/4/ip`

```
{
  "address": "**.***.***.***",
  "mask": "255.255.255.0",
  "mask_cidr": 24
}
```

Impact

This vulnerability allows an attacker to control outbound HTTP requests made by the Glances IP plugin via the public_api configuration parameter.

Server-Side Request Forgery (SSRF):
The application can be forced to send requests to arbitrary endpoints, including internal services and localhost.
Credential Leakage:
When public_username and public_password are configured, they are automatically sent in the Authorization: Basic header to any target defined in public_api, exposing credentials to attacker-controlled servers.
Internal Network Access:
The vulnerability enables access to internal resources such as:
127.0.0.1 (localhost services)
Private network ranges (192.168.x.x, 10.x.x.x, 172.16.x.x)
Cloud Metadata Exposure:
The application can be directed to query cloud metadata endpoints such as:
http://169.254.169.254/
potentially exposing sensitive credentials (e.g., IAM tokens in cloud environments)
Data Injection / Manipulation:
Responses from attacker-controlled servers are accepted and stored by Glances, then exposed via /api/4/ip, allowing injection of arbitrary data into the application.

NOTE

Vulnerability Location

The issue originates from how the public_api configuration value is handled and used without validation.

1. Source of user-controlled input

File: glances/plugins/ip/**init**.py
(around lines ~64–82)

```
self.public_api = self.get_conf_value("public_api", default=[None])[0]
self.public_username = self.get_conf_value("public_username", default=[None])[0]
self.public_password = self.get_conf_value("public_password", default=[None])[0]
public_api is fully user-controlled via configuration
```

No validation is applied at this stage

2. Missing validation before usage

```
self.public_disabled = ( self.get_conf_value('public_disabled', default='False')
[0].lower() != 'false' or self.public_api is None or self.public_field is None )
```

Only checks if the value is None
No validation of:

- URL scheme
- Hostname
- IP address range

3. Vulnerable sink (critical point)

```
self.public_ip_thread = ThreadPublicIpAddress( url=self.public_api,            # ← user-controlled input username=self.public_username, password=self.public_password, refresh_interval=self.public_address_refresh_interval, )
```

The user-controlled public_api is passed directly into a network request
This is the SSRF entry point

4. Unsafe HTTP execution

File: glances/globals.py
(around lines ~360+)

```
def urlopen_auth(url, username, password, timeout=3): return urlopen( Request( url,   # ← no validation at all headers={ 'Authorization': 'Basic ' + base64.b64encode(f'{username}:{password}'.encode()).decode() }, ), timeout=timeout, )
```

- Accepts any URL
- Sends request blindly
- Automatically attaches credentials to any destination
- Root Cause

A user-controlled configuration value (public_api) is passed directly into an HTTP request without validation of scheme or destination, resulting in SSRF and credential leakage.

Recommendation
The fix must be applied before the URL is used, specifically in the IP plugin (**init**.py).

1. Enforce scheme restrictions
Allow only:
http
https
Reject:
file://
gopher://
ftp://
any non-HTTP protocol

This prevents protocol abuse and local file access

2. Validate destination host
Resolve the hostname to an IP address
Check the resolved IP against restricted ranges

Block if the IP is:

Loopback → 127.0.0.0/8
Private → 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
Link-local → 169.254.0.0/16 (cloud metadata services)

This prevents:

Internal network probing
AWS/GCP/Azure metadata access
localhost abuse

3. Enforce validation before thread creation

The validation must occur before initializing:

ThreadPublicIpAddress(...)
If validation fails:
Disable the plugin
Do not send any request

4. Trust boundary clarification
urlopen_auth() is a low-level utility
It should not be responsible for validation

The caller (IP plugin) must ensure:

Only safe, external URLs are passed

Why This Fix Works
Scheme validation blocks protocol-based attacks
IP validation blocks internal and cloud targets
Combined, they eliminate the SSRF attack surface while preserving legitimate use cases (public IP APIs)

Severity

High

CVE ID

CVE-2026-35587

Weaknesses

▶ CWE-918

Credits

Venukamatchi Reporter