Understanding CVE-2026-29059: Unauthenticated Path Traversal in Windmill and Nextcloud Flow

Author: Synthex **Site:** denizhalil.com CVSS 7.5 HIGH **Date:** July 2026

INTRODUCTION

Modern enterprise automation relies heavily on unified workflow engines to connect disparate infrastructure components, manage background tasks, and streamline internal processes. Platforms such as **Windmill** and its integrations—including **Nextcloud Flow**—provide robust execution environments for scripts and automated jobs. However, because these systems process sensitive data and hold elevated permissions across networks, any inherent flaw severely expands an organization's attack surface. When central infrastructure automation platforms contain security flaws, the operational impact can extend far beyond a single application, potentially compromising connected services and host systems. Identified as **CVE-2026-29059**, a critical Unauthenticated Path Traversal vulnerability was discovered in versions of Windmill prior to `1.603.3`. This vulnerability allows remote, unauthenticated attackers to escape designated filesystem boundaries and retrieve arbitrary files from the underlying server host or container environment without requiring valid user credentials.

LEARNING OBJECTIVES

After reading this article, security engineers and system administrators will be able to:

- Understand the core mechanism and root cause of Path Traversal vulnerabilities (CWE-22).
- Analyze the vulnerability mechanics of CVE-2026-29059 within the endpoint architecture of Windmill and Nextcloud Flow.
- Identify the affected software versions, dependencies, and integration ecosystems.
- Evaluate potential downstream impacts, including sensitive data disclosure and potential privilege escalation.
- Apply actionable remediation techniques and secure coding practices to prevent similar path traversal issues.

WHAT IS WINDMILL/NEXTCLOUD FLOW < 1.603.3 - UNAUTHENTICATED PATH TRAVERSAL CVE-2026-29059

CVE-2026-29059 represents a critical security vulnerability categorized under **CWE-22: Improper Limitation of a Pathname to a Restricted Directory**, commonly known as Path Traversal or Directory Traversal. The underlying flaw resides within the web application's log retrieval mechanism, specifically

hosted at the endpoint `/api/w/{workspace}/jobs_u/get_log_file/{filename}`. Under intended operational parameters, this API endpoint allows workspace users and system administrators to inspect execution logs generated during automated job runs by supplying a standard log identifier or filename. However, the primary issue stems from an fundamental architecture design oversight regarding access control and input sanitization. Because this specific log endpoint does not enforce mandatory user authentication, it remains publicly reachable by any remote actor with network visibility to the instance. Furthermore, the application backend fails to validate, filter, or restrict relative path sequences supplied inside the `{filename}` URL path variable, allowing supplied input to directly influence local file system calls.

When an attacker constructs a request containing relative directory traversal sequences (such as `../`), the application concatenates this unvalidated input with its internal base log directory path. As a result, the filesystem resolves the path beyond the boundaries of the designated log directory, giving unauthenticated remote attackers the ability to traverse up to the host or container's root filesystem and arbitrary read protected system files.

- **Target Endpoint:** Unauthenticated access via `/api/w/{workspace}/jobs_u/get_log_file/{filename}`.
- **Vulnerability Mechanism:** Lack of input sanitization allowing `../` sequences to escape restricted directories.
- **Primary Threat:** Arbitrary read access to critical environment variables, configuration files, and system passwords.
- **Secondary Risk:** Credential extraction (e.g., `SUPERADMIN_SECRET`) leading directly to full privilege escalation or Remote Code Execution (RCE).

Vulnerable Source Code Example (Conceptual)

The following pseudo-code demonstrates how unvalidated path concatenation leads directly to CVE-2026-29059:

```
// Vulnerable API Route Handler (Rust / Actix / Axum pseudo-code)
async fn get_log_file(
    Path((workspace, filename)): Path<(String, String)>
) -> Result<NamedFile, Error> {
    // UNSECURE: Base path is concatenated directly with user-supplied filename
    // without sanitizing relative sequences like "../"
    let log_directory = format!("/tmp/windmill/workspaces/{}/logs", workspace);
    let target_path = PathBuf::from(log_directory).join(filename);

    // The system opens and serves whatever path target_path resolves to
    // e.g., /tmp/windmill/workspaces/default/logs/../../../../etc/passwd -> /etc/
passwd
    let file = NamedFile::open(target_path)?;
    Ok(file)
}
```

TECHNICAL DETAIL: HOW THE VULNERABILITY WORKS

The primary root cause of CVE-2026-29059 lies in the improper processing and concatenation of untrusted client input within Windmill's log routing logic. In modern web architectures, API endpoints handling file access typically enforce a strict separation between user-supplied identifiers and absolute server paths. However, the vulnerable `/api/w/{workspace}/jobs_u/get_log_file/{filename}` route accepts a dynamic `{filename}` parameter directly from the HTTP request URL without performing essential input filtering or sanitization against relative path characters. When a request arrives at this endpoint, the web framework extracts the `{filename}` string and appends it directly to a pre-configured base directory allocated for workspace logs. Because the application lacks both an initial authentication barrier and a path canonicalization step—such as resolving absolute paths via function calls like `canonicalize()`—the underlying operating system evaluates relative path sequences (such as `../`) literally. This architectural gap enables the request handler to navigate up the directory hierarchy, escaping the confines of the intended log folder and granting access to adjacent or system-level directories.

Furthermore, the cascading risk of this flaw extends significantly beyond standard file retrieval. Because the process running the application often holds access permissions to system environments and runtime configurations, an attacker can leverage this traversal to extract sensitive configuration files, runtime environment variables (e.g., `/proc/self/environ`), or internal database credentials. In environments where secret keys like `SUPERADMIN_SECRET` are exposed in memory or configuration files, extracting these credentials allows an attacker to authenticate as an administrator, effectively converting an arbitrary file read flaw into full Remote Code Execution (RCE).

Vulnerable Architectural Workflow:

- **Unauthenticated Request Ingestion:** An external client transmits an unauthenticated HTTP `GET` request containing relative directory traversal sequences (`../`) embedded inside the `{filename}` path parameter.
- **Direct Path Concatenation:** The backend API handler extracts the raw string from the URL and concatenates it directly with the internal workspace log storage directory without validating or stripping input patterns.
- **Canonical Boundary Escape:** The operating system resolves the relative path sequences, traversing out of the restricted log directory and anchoring the target path to critical host files (e.g., `/etc/passwd` or application environment files).
- **Unchecked File Streaming:** The application verifies file existence and opens an unbuffered read stream, returning the full contents of the targeted system file directly to the client inside the HTTP response body.

Example Request & Response Output:

```
GET /api/w/default/jobs_u/get_log_file/..%2f..%2f..%2f..%2fetc%2fpasswd HTTP/1.1
Host: target-windmill-instance.local
User-Agent: Mozilla/5.0
Accept: */*

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 1358

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...
```

AFFECTED SOFTWARE & PLUGINS

The overall blast radius of CVE-2026-29059 spans both standalone developer environments and integrated enterprise application ecosystems where Windmill is deployed as an execution microservice. Because Windmill frequently powers backend automation, background worker pipelines, and scheduled cron tasks, any installation running an unpatched core image exposes the entire host environment to unauthorized directory access. Organizations relying on third-party software bundles

or containerized stacks that integrate Windmill—such as Nextcloud Flow—inherit this path traversal risk directly if their underlying engine builds remain on vulnerable version branches.

- **Windmill Core Platform Engine:** All primary Windmill core instances and execution worker containers running versions prior to `1.603.3`. The official fix and canonical path validation checks are fully implemented starting in release **v1.603.3**.
- **Nextcloud Flow App Integration:** Nextcloud instances utilizing the Nextcloud Flow external app integration for workflow automation, specifically deployments running bundled Windmill engine versions lower than `1.603.3` or app container releases prior to `1.3.0`.
- **Embedded & Custom Microservice Deployments:** Third-party developer platforms, custom Docker Compose stacks, or Kubernetes Helm releases that embed the Windmill API container image without pinning backend dependencies to patched upstream releases.

CONCLUSION

CVE-2026-29059 serves as a stark reminder of the critical security risks inherent in modern workflow automation engines that expose unauthenticated endpoints. As platforms like Windmill and Nextcloud Flow become central hubs for orchestration, they naturally accumulate elevated privileges across organizational networks. Consequently, a single fundamental architectural flaw—such as omitting authentication checks on input-driven file operations—can severely expand an enterprise's overall attack surface and compromise its broader infrastructure stack. The mechanics of this vulnerability highlight why input handling must never be treated as a secondary defense layer. When applications process dynamic file paths without rigorous canonicalization, relative path sequences allow remote actors to easily bypass intended directory boundaries. In automation environments, where configuration files, memory environments, and local stores frequently hold sensitive credentials, an arbitrary file read vulnerability rarely remains an isolated read-only issue; it rapidly turns into a major pathway for identity theft and system takeover. To effectively mitigate the immediate threat posed by CVE-2026-29059, system administrators, DevOps teams, and security engineers must prioritize upgrading vulnerable installations. Windmill core instances must be updated to **version 1.603.3 or higher**, while integrated solutions like Nextcloud Flow should be moved to **release 1.3.0 or later**. In instances where immediate patching cannot be scheduled, security teams should implement defensive boundary controls, such as restricting access to the `/api/w/*/jobs_u/get_log_file/` route via reverse proxies or Web Application Firewalls (WAFs).

From a broader secure software development lifecycle (SDLC) perspective, preventing path traversal flaws requires adopting defensive engineering principles by design. Developers building file-handling APIs must enforce strict authentication requirements before routing requests to backend resources. Furthermore, all user-controlled input must undergo rigorous validation—combining strict character whitelisting with explicit path normalization checks to guarantee that resolved file targets remain strictly contained within designated system boundaries.