PLAINBIT

2026 월간 위협 분석 보고서

# n8n 워크플로우 자동화 플랫폼 원격 코드 실행 취약점 분석 (CVE-2025-68613)

2026. 02.

※ 본 보고서는 2026년 2월 국가사이버안보센터(NCSC) 합동분석협의체를 통해 발간되었습니다.

ⓒ 2025. Plainbit Co., Ltd. All rights reserved.

2026 월간 위협 분석 보고서

# n8n 워크플로우 자동화 플랫폼 원격 코드 실행 취약점 분석 (CVE-2025-68613)

## Contents

# [ 01 개요 ]

최근 기업 환경에서는 반복적인 업무 처리와 운영 효율성 향상을 목적으로 워크플로우 자동화 도구의 활용이 점차 확대되고 있다. 이러한 흐름 속에서 n8n은 오픈소스 기반의 자동화 플랫폼으로, 다양한 외부 서비스 및 내부 시스템과의 연계를 제공하고 비교적 자유로운 워크플로우를 설계할 수 있다는 점에서 여러 조직에서 활용되고 있다. 그러나 자동화 범위가 확대될수록 내부 로직 실행 구조와 데이터 처리 방식이 공격 표면으로 작용할 가능성 또한 함께 증가하고 있다.

n8n은 워크플로우 실행 과정에서 사용자가 정의한 표현식(Expression)을 통해 데이터 가공, 조건 분기, 동적 값 처리를 수행한다. 해당 표현식은 자바 스크립트 런타임 환경에서 실행되며 노드 간 데이터 전달과 실행 흐름을 제어하는 핵심 요소로 사용된다. 이와 같은 구조는 자동화를 유연하게 구현할 수 있도록 하지만, 표현식 처리 과정에 대한 보안 통제가 미흡할 경우 공격자가 의도적으로 조작된 입력 값을 통해 내부 실행 컨텍스트를 벗어나 시스템 정보 노출 및 임의 코드 실행으로 이어질 수 있는 위험성을 내포하고 있다.

실제로 최근 n8n에서 표현식 처리 로직과 관련된 보안 취약점이 공개되었으며, 해당 취약점을 악용할 경우 원격 코드 실행으로 확장될 수 있다는 것이 확인되었다. 이는 단순한 기능 오남용 수준을 넘어 n8n이 설치된 서버 자체에 대한 침해로 이어질 수 있는 위협 시나리오를 의미한다. 특히 n8n이 내부 시스템 접근 권한을 가진 상태로 운영되는 경우, 공격자는 자동화 플랫폼을 경유하여 내부 네트워크 및 연계 시스템으로 추가 공격을 수행할 가능성이 존재한다.

본 보고서는 n8n 환경에서 발생할 수 있는 이러한 보안 위협을 구조적·기술적 관점에서 분석하고, 침해사고 분석 및 대응 관점에서 활용 가능한 정보를 제공하는 것을 목적으로 한다. 단순히 취약점의 존재 여부를 설명하는 데 그치지 않고 공격자가 해당 취약점을 어떤 방식으로 악용할 수 있는지 그리고 그 과정에서 시스템 내부에 어떤 흔적이 남는지를 중점적으로 다룬다. 이를 통해 n8n을 운영 중인 환경에서 이상 행위를 식별하고 분석하는 데 필요한 실질적인 관점을 제시하고자 한다.

# [ 02 n8n 워크플로우 자동화 플랫폼 ]

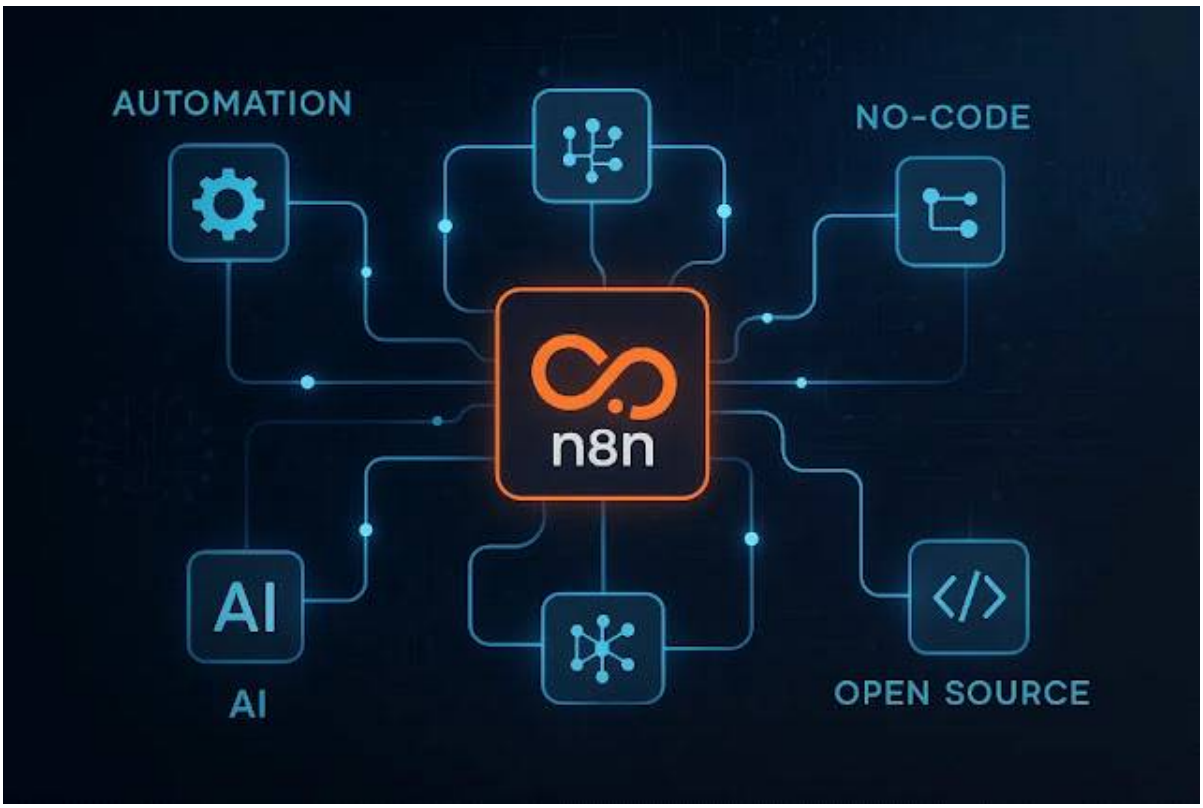
## 1. n8n이란?

n8n은 다양한 애플리케이션과 서비스를 연결하여 업무 흐름을 자동화할 수 있도록 지원하는 오픈소스 기반의 워크플로우 자동화 플랫폼이다. 사용자는 별도의 복잡한 개발 없이도 시각적 인터페이스를 통해 데이터 처리, API 연동, 이벤트 기반 작업 등을 구성할 수 있으며, 필요에 따라 사용자 정의 로직을 직접 작성하는 것도 가능하다.

n8n은 로우코드(Low-code) 및 노코드(No-code) 환경을 지향하지만, 단순 자동화 도구에 그치지 않고 자바 스크립트 기반의 사용자 정의 코드 실행 기능을 함께 제공한다는 점에서 기술적 유연성이 매우 높은 플랫폼으로 분류된다. 이러한 특성으로 인해 n8n은 IT 운영 자동화, DevOps 파이프라인 구성, 데이터 처리 자동화, API 오케스트레이션, AI 기반 워크플로우 구축 등 다양한 목적으로 활용되고 있다.

또한, n8n은 자체 호스팅 방식과 클라우드 서비스 형태를 모두 지원한다. 조직은 이를 통해 인프라와 데이터에 대한 통제권을 유지한 채 자동화 환경을 구축할 수 있으며, 내부 시스템·외부 SaaS·클라우드 서비스 간 연계를 중앙에서 관리할 수 있다. 이로 인해 n8n은 단순한 자동화 도구를 넘어 조직의 핵심 업무 흐름을 담당하는 중앙 오케스트레이션 계층으로 기능하는 경우가 많다.

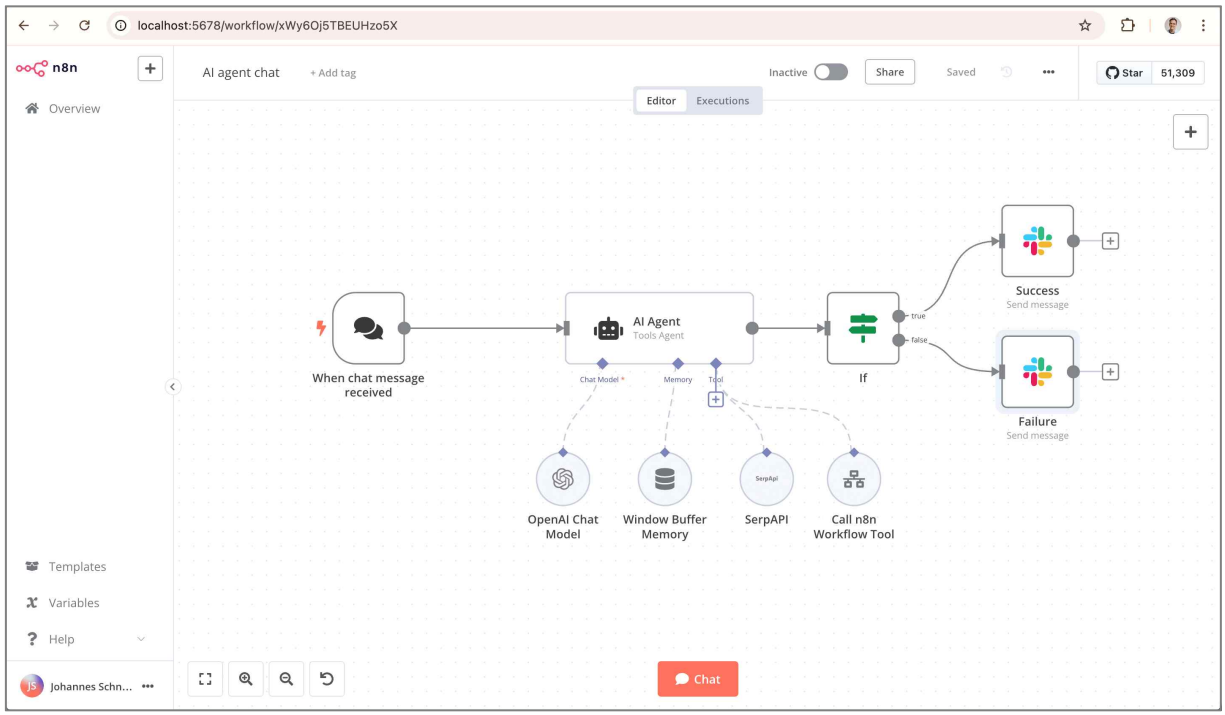
이와 같이 n8n은 업무 효율성과 확장성을 동시에 제공하는 장점이 있으나, 자동화 플랫폼 특성상 다수의 시스템 접근 권한과 민감한 인증 정보를 다루게 되므로 보안 취약점 발생 시 그 영향 범위가 조직 전반으로 확산될 수 있는 구조적 특징을 가진다.




▲ n8n 자동화 플랫폼

## 2. 워크플로우 구조 및 동작 방식

n8n의 자동화 로직은 '워크플로우(Workflow)'라는 단위로 구성되며, 각 워크플로우는 트리거 노드와 여러 개의 작업 노드(Node)가 실행 순서를 따라 연결된 방향을 가진 그래프 구조로 구성된다. 트리거 노드는 워크플로우 실행 조건을 정의하며, 예약 실행, 웹훅 호출, 외부 이벤트 수신 등 다양한 방식이 존재한다.




▲ n8n 워크플로우
(출처 : n8n 공식 깃허브, https://github.com/n8n-io/n8n)

트리거 노드가 활성화되면 n8n 엔진은 연결된 노드들을 순차적으로 실행한다. 각 노드는 입력 데이터를 전달받아 정의된 작업을 수행한 후, 그 결과를 다음 노드로 전달한다. 워크플로우 실행 과정에서 각 노드의 입력 값과 출력 값은 로그로 기록되며, 사용자는 이를 통해 실행 결과를 확인하고 디버깅할 수 있다.

n8n은 내부적으로 Node.js 기반 런타임 환경에서 동작하며, 워크플로우 실행 시 사용자가 입력한 표현식을 서버에서 직접 실행하여 그 결과를 계산한다. 이는 사용자의 화면에서 단순히 값을 계산하는 방식이 아니라 실제로 n8n 서버가 해당 코드를 대신 실행해주는 구조를 의미한다. 이러한 방식은 자동화 로직을 유연하게 구성할 수 있다는 장점이 있지만, 입력 값이 충분히 제한되지 않을 경우 서버가 의도하지 않은 동작까지 수행하게 될 수 있어 보안 상 주의가 필요하다.

## 3. 표현식(Expression) 기능

n8n의 '표현식(Expression)' 기능은 워크플로우 실행 과정에서 값을 동적으로 계산하거나, 이전 노드의 결과를 참조하여 입력 값을 구성하기 위해 제공되는 핵심 기능이다.. 사용자는 이중 중괄호(`{{ }}`)로 감싸진 형태의 표현식을 입력해 단순한 정적 값 대신 실행 지점에 계산되는 값을 노드 설정에 사용할 수 있다.

예를 들어, 다음과 같은 표현식은 단순한 산술 계산을 수행한다.

```
{{ 1 + 1 }}
```

워크플로우가 실행되면 n8n 서버는 해당 표현식을 계산하여 결과 값인 '2'로 치환한다. 이와 같은 방식으로 표현식은 숫자 계산, 문자열 결합 등 기본적인 연산에 활용될 수 있다.

또한, 표현식은 워크플로우 실행 과정에서 생성되는 데이터에 접근할 수 있다. 다음 예시는 이전 노드의 출력 데이터 중 'userId' 값을 참조하는 표현식이다.

```
{{ $json.userId }}
```

이 경우 표현식은 워크플로우 데이터 컨텍스트 내에 포함된 값을 참조해 동적으로 입력 값을 구성한다. 이러한 기능은 n8n 자동화 로직을 유연하게 구성하는데 필수적인 요소이다.

중요한 점은 n8n의 표현식이 단순한 템플릿 언어나 제한된 수식 전용 문법이 아니라 자바 스크립트 문법을 그대로 기반해 동작한다는 점이다. 따라서, 산술 연산이나 데이터 참조 뿐만 아니라 자바 스크립트에서 사용 가능한 다양한 구문을 표현식 내부에서 그대로 사용할 수 있다.

이로 인해 표현식 내부에서는 자바 스크립트의 'this' 구문 역시 사용할 수 있다. 일반적인 자바 스크립트 환경에서의 this 구문을 실행하면 다음과 같다.

**<자바 스크립트 실행 예시>**

```
function showThis() {
  return this;
}

console.log(showThis());
```

**실행결과 :**

```
global { process, setTimeout,... }
```

위 코드는 Node.js 환경에서 실행될 경우, 함수 호출 방식에 따라 전역 객체를 반환할 수 있으며 실행 결과는 Node.js 전역 객체의 일부를 포함한다. 이는 자바 스크립트 코드가 Node.js 런타임의 전역 스코프에서 실행될 때 this가 전역 객체를 가리킬 수 있음을 보여주는 일반적인 예시이다.

다음은 동일한 개념을 n8n 표현식 환경에서 확인하기 위한 예시이다. 다음 표현식은 즉시 실행 함수를 통해 현재 실행 컨텍스트의 this를 반환한다. 취약한 n8n 버전의 표현식 실행 환경에서는 이 코드가 일반적인 자바 스크립트 예제와 유사하게 Node.js 전역 객체를 반환했으며, 실행 결과는 다음과 같은 형태를 가진다.

**<n8n 표현식>**

```
{{ (function () { return this })() }}
```

**실행결과 :**

```
{  process: {...},
  Buffer: {...},
  setTimeout: [Function],
  ...
}
```

중요한 차이점은 일반적인 자바 스크립트 코드에서는 이러한 전역 객체 접근이 개발자가 인지한 상태에서 이루어지는 반면, n8n 표현식의 경우 표현식이 단순한 데이터 처리 기능으로 인식되는 상황에서도 동일한 전역 객체 접근이 가능했다는 점이다. 이로 인해 표현식이 의도된 사용 범위를 넘어 서버 실행 환경과 직접 상호작용할 수 있는 구조가 형성되었다.

본 보고서에서 다루는 CVE-2025-68613 취약점은 바로 이러한 표현식 실행 구조의 특성을 악용한 사례로, 표현식이 실행되는 컨텍스트가 충분히 격리되지 않았을 때 어떤 보안 문제가 발생할 수 있는지를 보여준다. 다음 장에서는 표현식이 실제로 어떤 방식으로 해석되어 실행되고, 그 과정에서 보안 요소가 어떻게 우회되었는지를 기술적인 관점에서 상세히 분석한다.

# [ 03 CVE-2025-68613 취약점 분석 ]

## 1. 취약점 개요

CVE-2025-68613 취약점은 n8n의 표현식(Expression) 기능이 서버에서 실행되는 구조적 특성과 맞물려 발생한 원격 코드 실행 취약점이다. 앞서 살펴본 바와 같이 n8n 표현식은 워크플로우 실행 시 서버의 Node.js 런타임 환경에서 해석되어 실행되며, 이 과정에서 실행 범위가 충분히 제한되지 않을 경우 의도하지 않은 동작으로 이어질 수 있다.

해당 취약점은 인증된 사용자가 워크플로우 내에서 자바 스크립트 표현식을 입력할 수 있도록 설계된 정상적인 기능이 공격에 악용될 수 있는 구조에서 비롯된다. 표현식이 단순한 값 계산이나 데이터 처리 목적을 넘어 실행 환경에 접근할 수 있게 되면서, 결과적으로 n8n 프로세스 권한으로 임의 코드 실행이 가능해진다.

NVD(National Vulnerailbity Database)에 등록된 정보에 따르면, CVE-2025-68613은 CVSS 점수 8.8로 심각도는 높음(High)으로 평가되었다. 이는 공격 난이도가 낮고, 취약점이 악용될 경우 서버 및 시스템 전반에 중대한 영향을 미칠 수 있음을 의미한다.

### (1) 영향받는 버전 및 수정된 버전

본 취약점은 특정 설정이나 확장 기능에 국한된 문제가 아니라 표현식 기능이 포함된 n8n의 다수 버전에 영향을 미친다. 공식 권고사항 및 패치 릴리즈에 따르면 영향받는 버전과 수정된 버전은 다음과 같다.

| 구분 | 버전 범위 |
|---|---|
| 영향받는 버전 | ▪ 0.211.0 이상 1.120.4 이전 버전 / 1.121.0 이상 1.220 이전 버전 |
| 수정된 버전 | ▪ 1.120.4 버전, 1.121.1 버전, 1.122.0 이상 버전 |

### (1) 공격 전제 조건 및 특성

본 취약점은 외부 공격자가 인증 없이 즉시 서버를 장악하는 형태라기보다는 유효한 n8n 계정을 보유한 사용자에 의해 악용될 가능성이 높다. 특히 워크플로우를 생성하거나 편집할 수 있는 권한이 있는 경우 관리자 권한이 없더라도 취약점 악용이 가능하다. 이로 인해 다중 사용자 환경이나 조직 내부에서 여러 명이 n8n을 공동으로 사용하는 경우, 계정 탈취 또는 내부자 계정 악용 시 보안 위험이 크게 증가할 수 있다.

### (2) 잠재적 위험 및 영향 범위

n8n은 외부 SaaS, 내부 시스템, 데이터베이스 등과 연계되어 자동화된 작업을 수행하는 경우가 많으며, 이 과정에서 다양한 인증 정보와 민감 데이터를 처리한다. 이러한 환경에서 본 취약점이 악용될 경우, 단일 서버 침해에 그치지 않고 자동화 플랫폼을 경유한 추가 피해로 이어질 가능성이 존재한다.

대표적으로 예상되는 위험은 다음과 같다.

- 서버에서의 임의 명령 실행을 통한 시스템 통제 권한 획득
- 워크플로우에 저장되거나 참조되는 인증 정보 및 민감 데이터 노출
- 자동화 로직 변조를 통한 업무 흐름 변조 및 비정상 작업 수행
- n8n과 연계된 외부 서비스 또는 내부 시스템으로의 추가 침투

CVE-2025-68613 취약점은 표현식 기능이라는 정상적인 활용 방법을 통해 발생하는 취약점으로, n8n이 조직의 핵심 자동화 도구로 활용될수록 그 영향 범위와 실제 피해 규모가 확대될 수 있다.

## 2. 취약점 동작 방식

본 절에서는 CVE-2025-68613 취약점이 실제로 어떤 흐름을 통해 동작하는지를 단계적으로 설명한다. 위 2.3절에서 살펴본 n8n 표현식 기능의 특성을 바탕으로 표현식이 입력·저장·실행되는 과정에서 어떻게 서버 측 코드 실행으로 이어질 수 있는지를 동작 관점에서 정리한다.

### (1) 워크플로우 표현식 실행 구조

n8n의 사용자가 워크플로우 내부에 자바 스크립트 표현식을 삽입하여 데이터를 동적으로 처리할 수 있도록 설계되어 있다. 이러한 표현식은 `{{ ... }}` 형태로 정의되며, 워크플로우 실행 시 서버의 Node.js 런타임 환경에서 실행된다.

설계상 표현식의 동작 범위는 워크플로우 데이터 처리에 제한되어야 하며, 다음과 같은 특성이 전제된다.

- 표현식은 워크플로우 데이터에만 적용되어야 함
- 운영체제 자원에 대한 접근 불가해야 함
- Node.js 내부 구조나 전역 객체 접근 불가해야 함
- 실행 결과만 안전하게 워크플로우로 반환해야 함

그러나 취약한 n8n 버전에서는 표현식이 별도의 샌드박스 없이 n8n 엔진과 동일한 실행 컨텍스트에서 실행되었다. 이로 인해 사용자 입력으로 전달된 자바 스크립트 코드가 워크플로우 데이터 범위를 넘어 서버 실행 환경에 접근할 수 있는 상태가 되었다. 즉, 표현식과 서버 코드 사이에 존재해야 할 신뢰 경계가 사실상 없는 구조이다.

### (2) 표현식 삽입

표현식이 신뢰할 수 없는 입력으로 취급되지 않고 그대로 자바 스크립트 코드로 실행되기 때문에 공격자는 단순한 데이터 계산이 아닌 실행 가능한 로직을 표현식에 직접 삽입할 수 있다.

다음은 공격자가 사용할 수 있는 악성 표현식의 예시이다.

```
{{ require('child_process').execSync('id') }}
```

해당 표현식은 워크플로우 실행 시 Node.js에 의해 직접 실행되며, `require()` 함수를 통해 내부 모듈을 로드한 뒤 운영체제 명령어를 실행한다.

이와 같은 공격이 가능한 이유는 다음과 같다.

- `require()` 함수가 표현식 실행 컨텍스트에서 사용 가능
- Node.js 핵심 모듈에 대한 접근이 충분히 제한되지 않음
- 표현식 실행 엔진이 엄격한 샌드박스 정책을 적용하지 않음
- 사용자 제공 표현식이 n8n 서비스와 동일한 권한으로 실행됨

이로 인해 공격자가 제어하는 입력 값이 신뢰된 서버 측 코드로 실행되는 구조가 형성된다.

### (3) 코드 실행 내부 동작 분석

공격자가 삽입한 임의 명령어를 포함한 표현식이 실행될 경우, 내부적으로 다음과 같은 단계가 수행된다.

#### 모듈 로딩

공격자가 삽입한 임의 명령어를 포함한 표현식이 실행되면, 먼저 Node.js의 내장 모듈인 `child_process`가 로드된다. `child_process` 모듈은 운영체제 수준에서 새로운 프로세스를 생성하거나 외부 명령을 실행하기 위해 제공되는 기능으로, 일반적으로는 서버 애플리케이션 내부의 신뢰된 코드에서만 사용되어야 한다. 그러나 취약한 n8n 표현식 실행 환경에서는 이러한 제한이 적용되지 않아 표현식 내부에서도 해당 모듈을 직접 로드할 수 있는 상태가 된다.

```
require('child_process')
```

#### 명령어 삽입 및 실행

`child_process` 모듈이 로드된 이후, 공격자는 `execSync()` 함수를 호출해 운영체제 명령어를 실행할 수 있다. `execSync()` 함수는 Node.js에서 제공하는 동기식 명령 실행 함수로, 지정된 명령어를 호스트 운영체제에서 직접 실행하고 그 결과를 표준 출력 형태로 반환한다.

```
execSync('id')
```

취약한 n8n 환경에서는 이 함수가 n8n 서비스 프로세스와 동일한 권한으로 실행되므로, 명령어는 n8n이 동작 중인 운영체제 사용자 권한을 그대로 상속받아 수행된다. 또한 `execSync()`는 동기 방식으로 동작하기 때문에 명령어 실행이 완료될 때까지 n8n 프로세스의 실행 흐름이 일시적으로 차단되며, 실행 결과는 자바 스크립트 값으로 반환되어 다시 워크플로우 실행 결과에 포함된다.

예를 들어 id 명령어가 실행될 경우, 반환되는 출력 값은 다음과 같다.

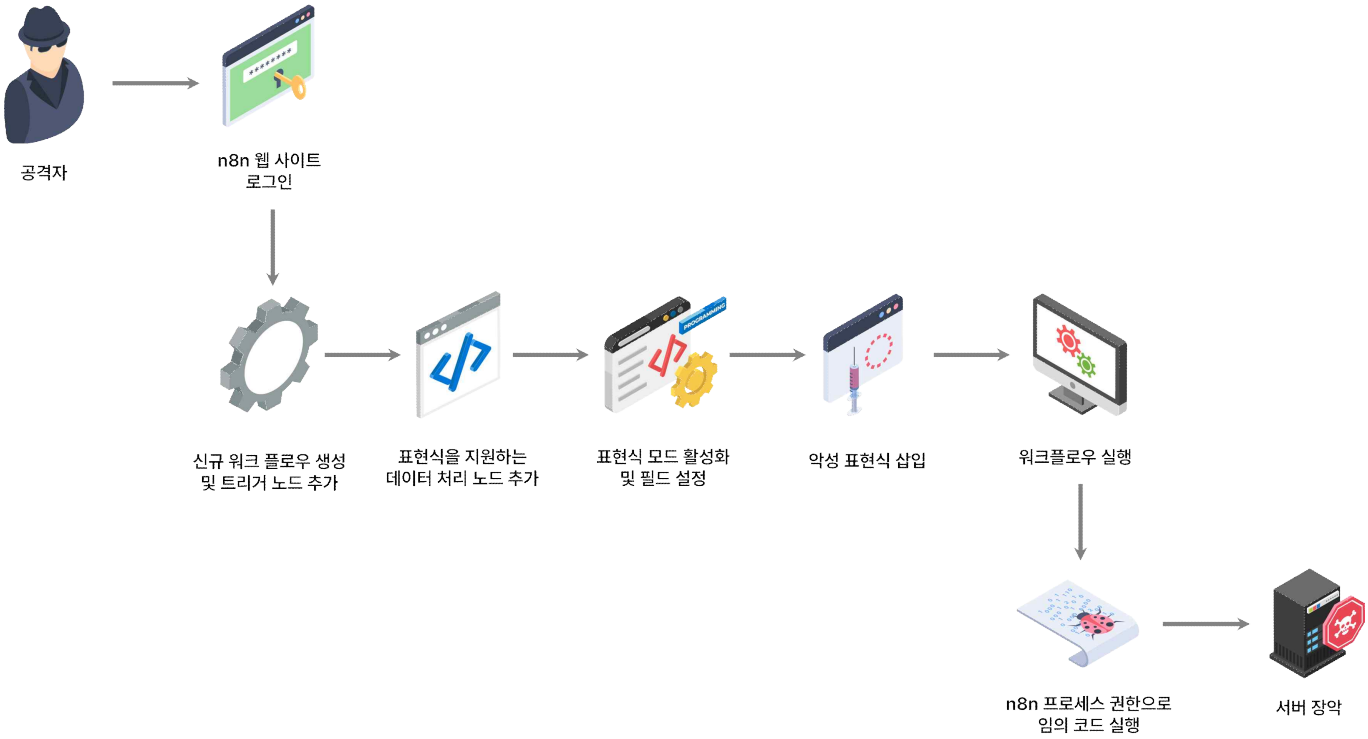
```
uid=1000(n8n) gid=1000(n8n) groups=1000(n8n)
```

이 출력은 공격자가 서버 상에서 임의의 명령을 실행할 수 있으며, 해당 명령이 n8n 프로세스의 운영체제 권한으로 수행되고 있음을 보여준다. 즉, 표현식 하나만으로 서버 내부에서 명령어 실행과 그 결과 수집까지 가능한 상태가 성립하게 된다.

이러한 구조적 결함으로 인해 n8n의 표현식 기능은 의도된 자동화 범위를 넘어 서버 전체를 제어할 수 있는 공격 경로로 악용될 수 있다. 다음 절에서는 이러한 동작 방식이 실제 공격 시나리오에서 어떻게 활용되는지를 살펴본다.

## 3. 공격 시나리오 및 재현

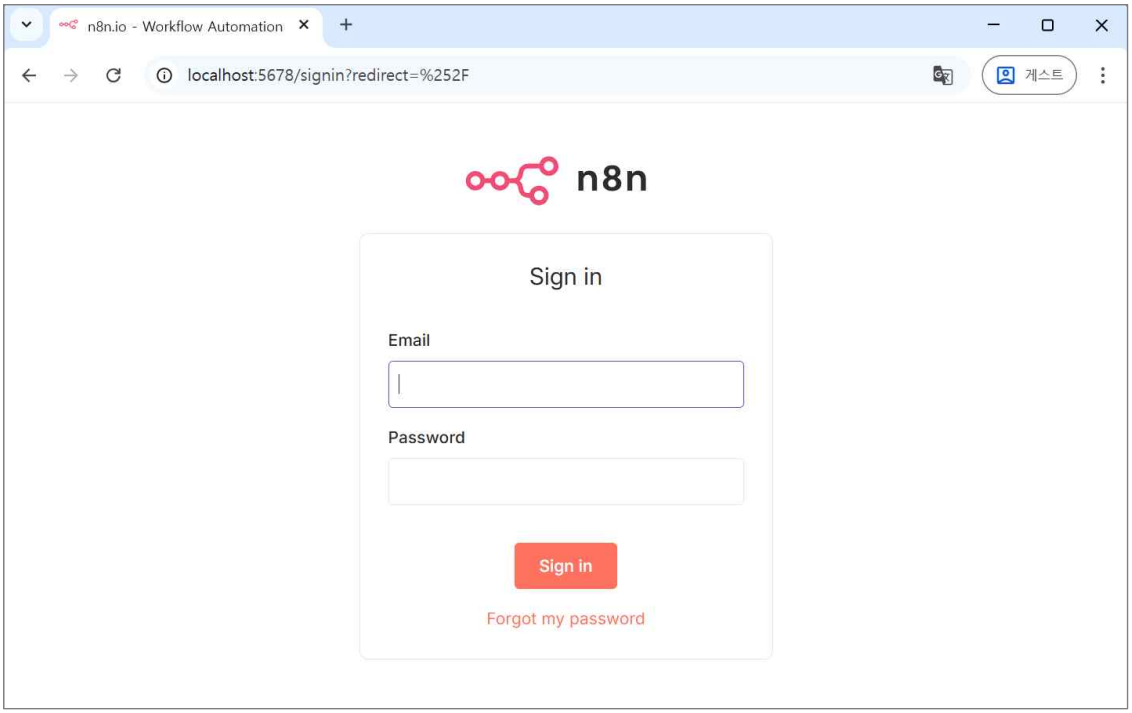
본 절에서는 CVE-2025-68613 취약점이 실제 환경에서 어떤 방식으로 악용될 수 있는지를 단계별 공격 시나리오 형태로 설명한다. 해당 시나리오는 관리자 권한 없이도 워크플로우를 생성하거나 편집할 수 있는 일반 사용자 권한만으로 재현이 가능하다는 점에서 실질적인 위협을 가진다.




▲ n8n 원격 코드 실행 취약점(CVE-2025-68613) 공격 흐름도

### (1) Step 1 : n8n 웹 접근 및 로그인

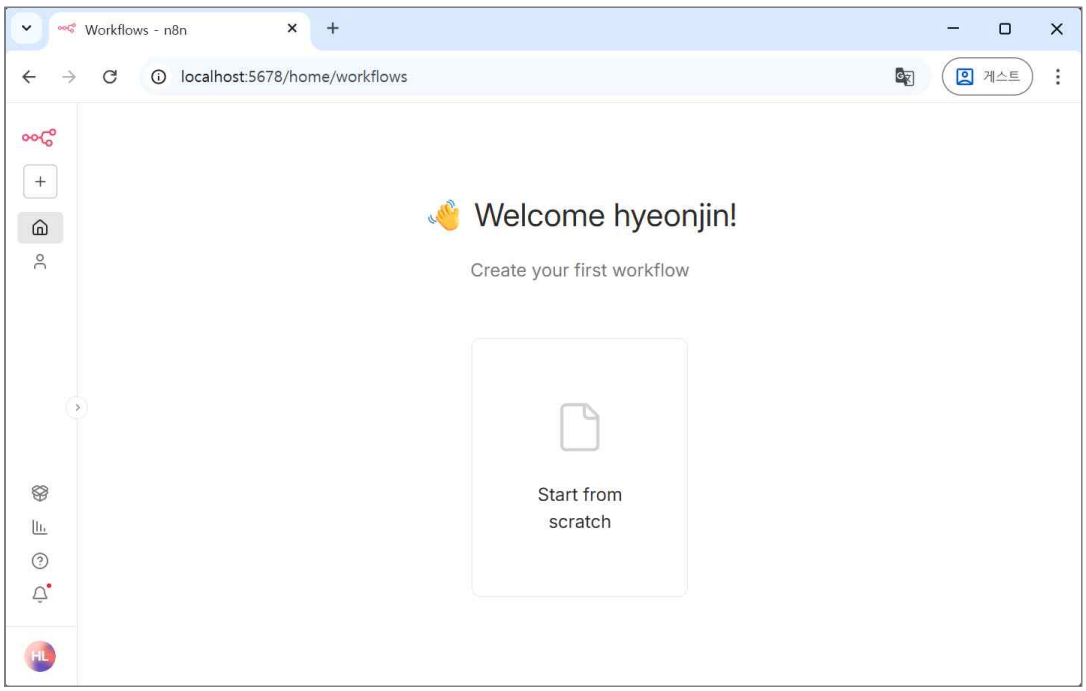
공격자는 먼저 유효한 n8n 계정을 사용해 웹 인터페이스에 로그인한다. 이때 관리자 권한은 필요하지 않으며, 워크플로우를 생성하거나 편집할 수 있는 일반 사용자 권한만 있으면 충분하다. n8n은 이메일을 기반으로 계정을 초대해 인증하는 방식을 사용하므로 외부 공격자가 임의로 신규 계정을 생성해 취약점을 악용하는 시나리오는 현실적으로 제한적이다. 다만 워크플로우 생성 또는 편집 권한을 보유한 내부 사용자, 혹은 계정 탈취를 통해 합법적인 접근 권한을 획득한 공격자에 의해 해당 취약점이 악용될 가능성은 충분히 존재한다. 특히 다중 사용자 환경이나 조직 내부에서 n8n을 공동으로 사용하는 경우, 내부자 위협 또는 계정 탈취로 인한 위험도가 크게 증가할 수 있다.




▲ n8n 로그인 페이지

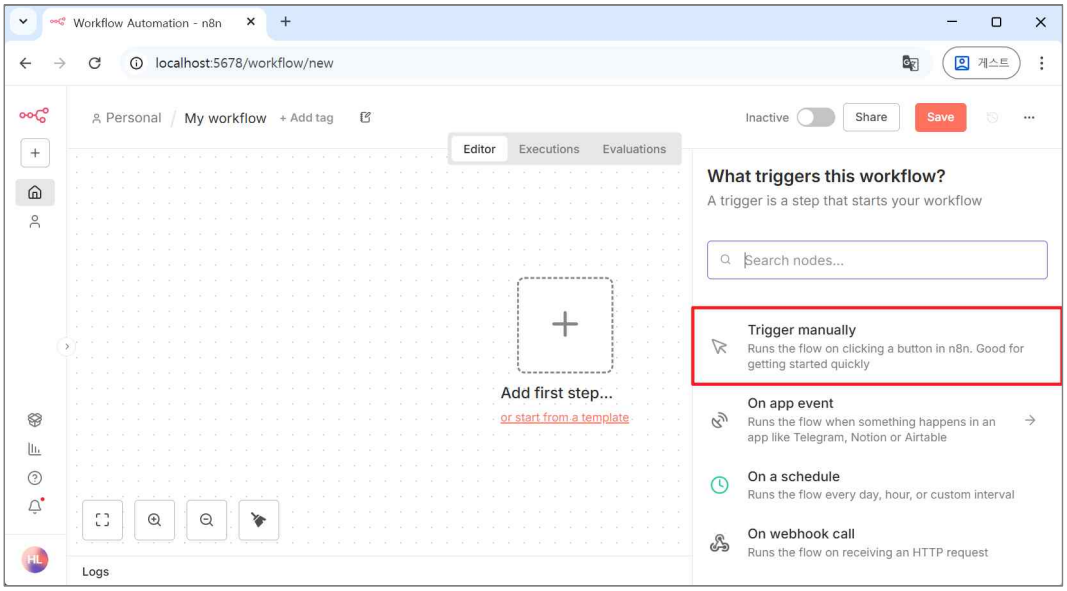
### (2) Step 2 : 신규 워크플로우 생성 및 트리거 노드 추가

로그인 이후 공격자는 신규 워크플로우를 생성한다. 기존 워크플로우를 수정할 필요 없이 기본 설정 상태의 빈 워크플로우만으로도 공격이 가능하다.




▲ 신규 워크플로우 추가 화면

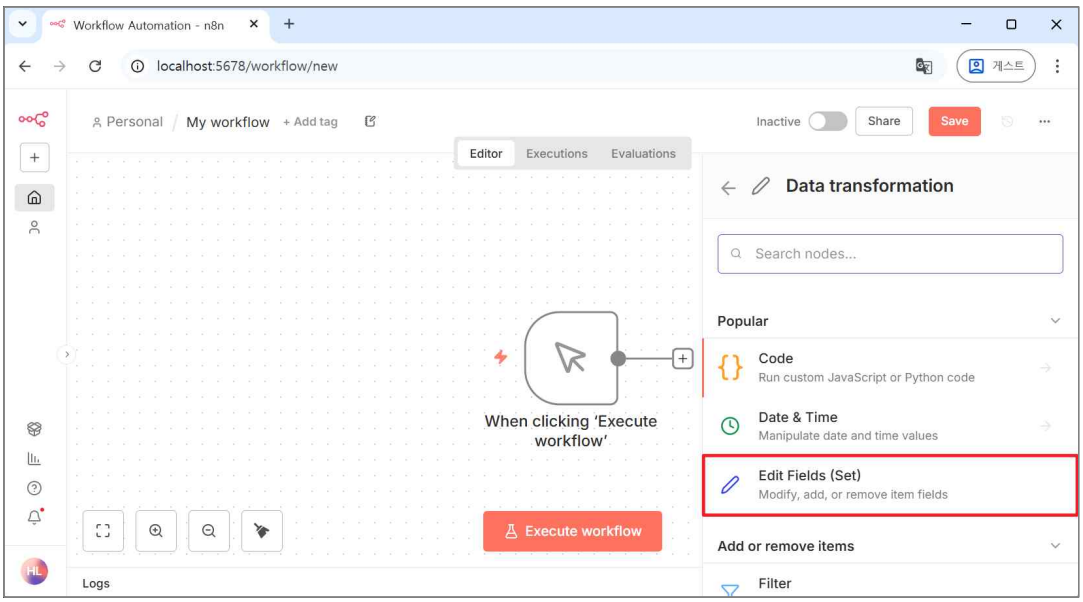
공격자는 워크플로우의 첫 번째 노드로 수동 트리거(Manual Trigger)를 추가한다. 수동 트리거는 사용자가 원하는 시점에 워크플로우를 즉시 실행할 수 있도록 하는 기본 기능이다. 해당 취약점은 워크플로우의 배포 과정이나 임포트 과정에서 발생하는 것이 아닌 일반적인 워크플로우 실행 과정 중에 발생하기 때문에 정상적인 워크플로우 실행 흐름 자체가 공격 트리거로 활용된다.




▲ 워크플로우 내 수동 트리거 추가

### (3) Step 3 : 표현식을 지원하는 데이터 처리 노드 추가

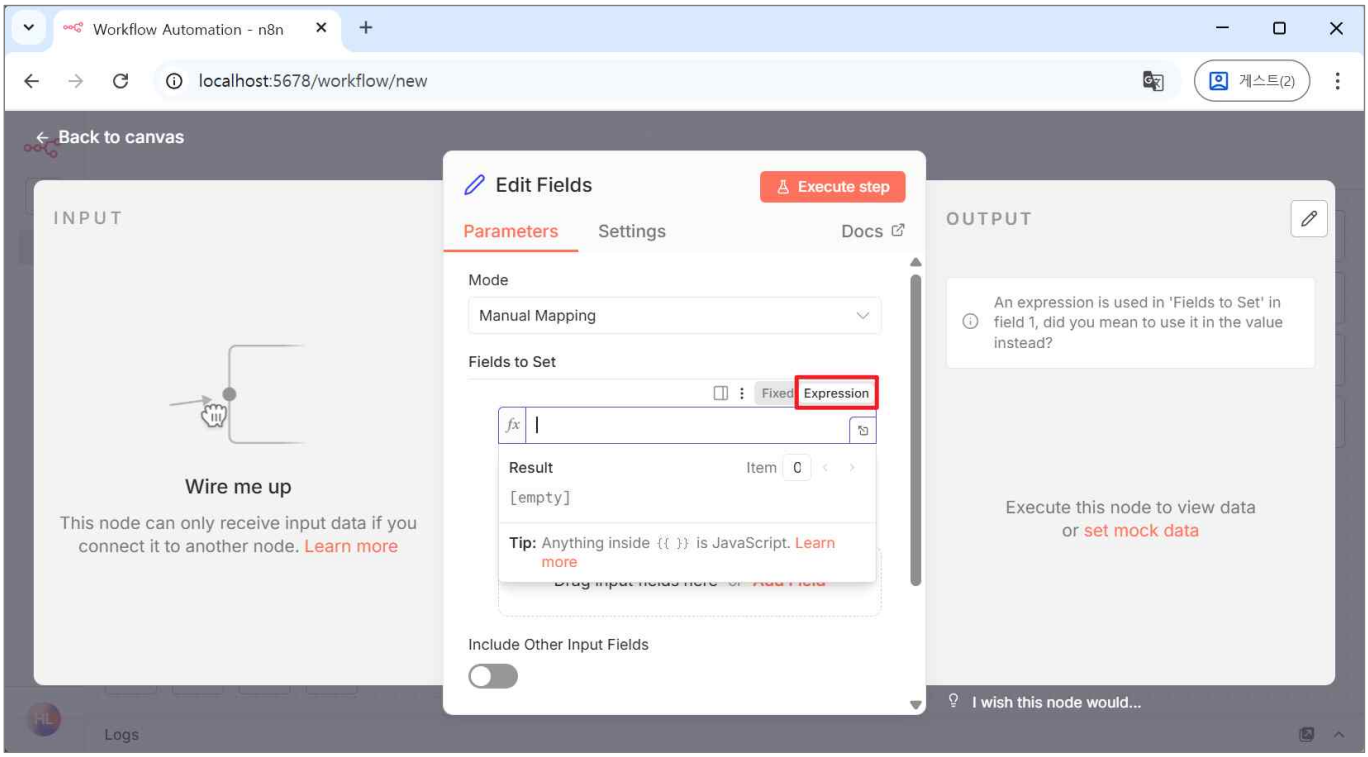
다음 단계로 공격자는 수동 트리거 뒤에 데이터 처리용 노드를 추가한다. 일반적으로 사용되는 Set 노드는 필드 값에 표현식(Expression)을 입력할 수 있는 기능을 제공하며, 본 취약점의 주요 공격 표면을 형성한다. Set 노드는 문자열, 숫자 등 다양한 필드 타입을 지원하며 각 필드 값에 대한 표현식 모드를 활성화할 수 있다. 이 표현식 모드는 단순한 정적 데이터가 아닌 서버 측에서 실행되면 자바 스크립트 코드로 처리하도록 한다.




▲ 'Edit Fields (Set)' 노드 추가

### (3) Step 4 : 표현식 모드 활성화 및 필드 설정

공격자는 Set 노드에서 새로운 문자열 필드를 추가한 후 해당 필드의 값을 표현식 모드로 전환한다. 이 과정에서 n8n은 입력된 값을 단순 문자열이 아닌 자바 스크립트 표현식으로 인식하게 된다. 이 단계에서까지 설정은 모두 n8n에서 제공하는 정상적인 기능 범위 내에서 이루어지며 별도 보안 경고나 제한 없이 수행된다.




▲ Set 노드에서 표현식 모드 활성화

### (4) Step 5 : 악성 표현식 삽입

표현식 입력 필드에 다음과 같은 악성 자바 스크립트 표현식을 삽입한다.

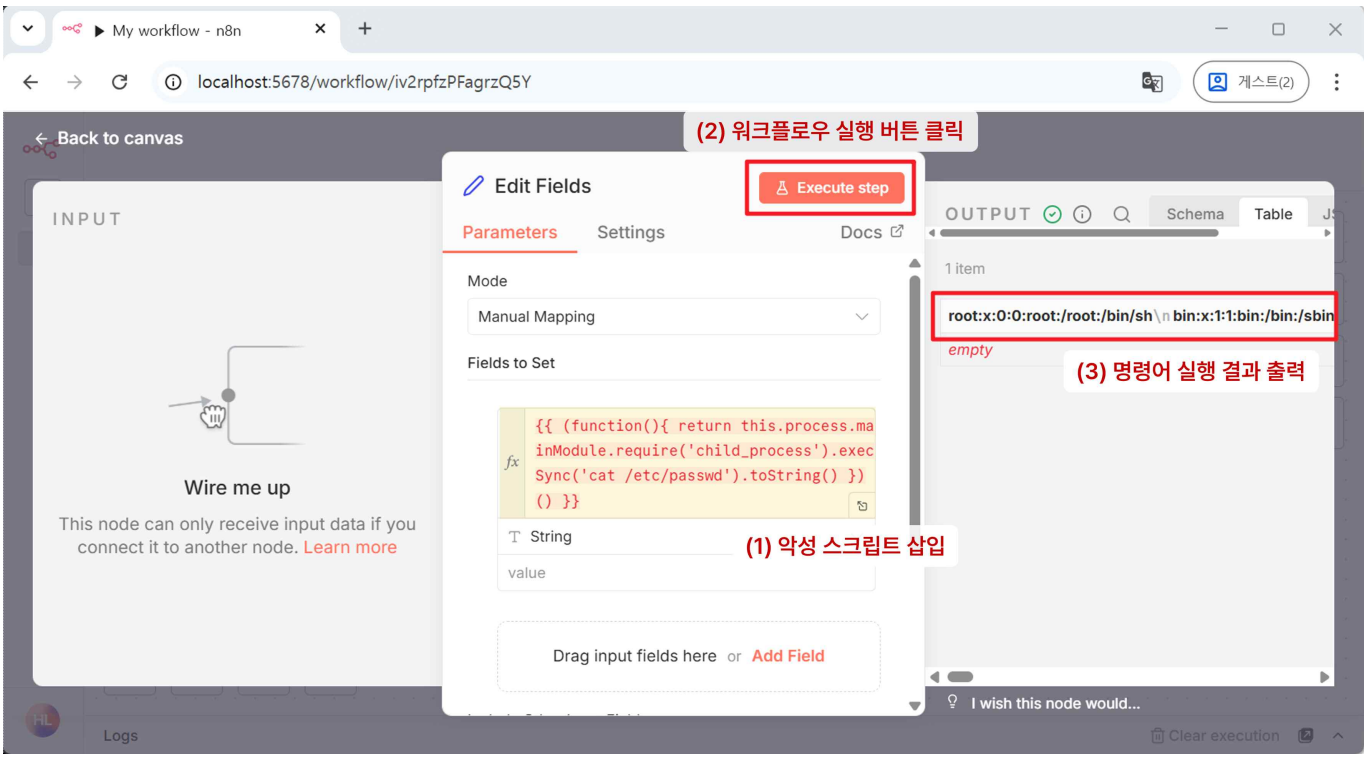
```
{{ (function(){ return this.process.mainModule.require('child_process').execSync('cat /etc/passwd').toString() })() }}
```

해당 표현식은 워크플로우 실행 시 n8n 표현식 엔진에 의해 서버 측에서 실행된다. 표현식 내부에서는 자바 스크립트의 `this` 구문을 통해 Node.js 전역 객체에 접근한 뒤 `process.mainModule.require()`를 사용해 `child_process` 모듈을 로드하고 운영체제 명령어를 실행한다.

이 과정에서 표현식은 데이터로 처리되지 않고 코드로 실행되며, 표현식 실행 컨텍스트가 격리되지 않은 구조로 인해 공격자는 의도된 자동화 범위를 벗어나 서버 운영체제에 직접 명령을 전달할 수 있게 된다.

### (5) Step 6 : 워크플로우 실행 및 결과 확인

공격자는 워크플로우 실행 버튼을 클릭해 수동 트리거를 실행한다. 워크플로우가 실행되면 Set 노드에 삽입된 표현식이 서버에서 실행되며 운영체제 명령어가 즉시 실행된다. 명령어 실행 결과는 자바 스크립트 문자열로 반환되어 Set 노드의 출력 값에 포함되며, 공격자는 웹 인터페이스를 통해 해당 결과를 직접 확인할 수 있다.



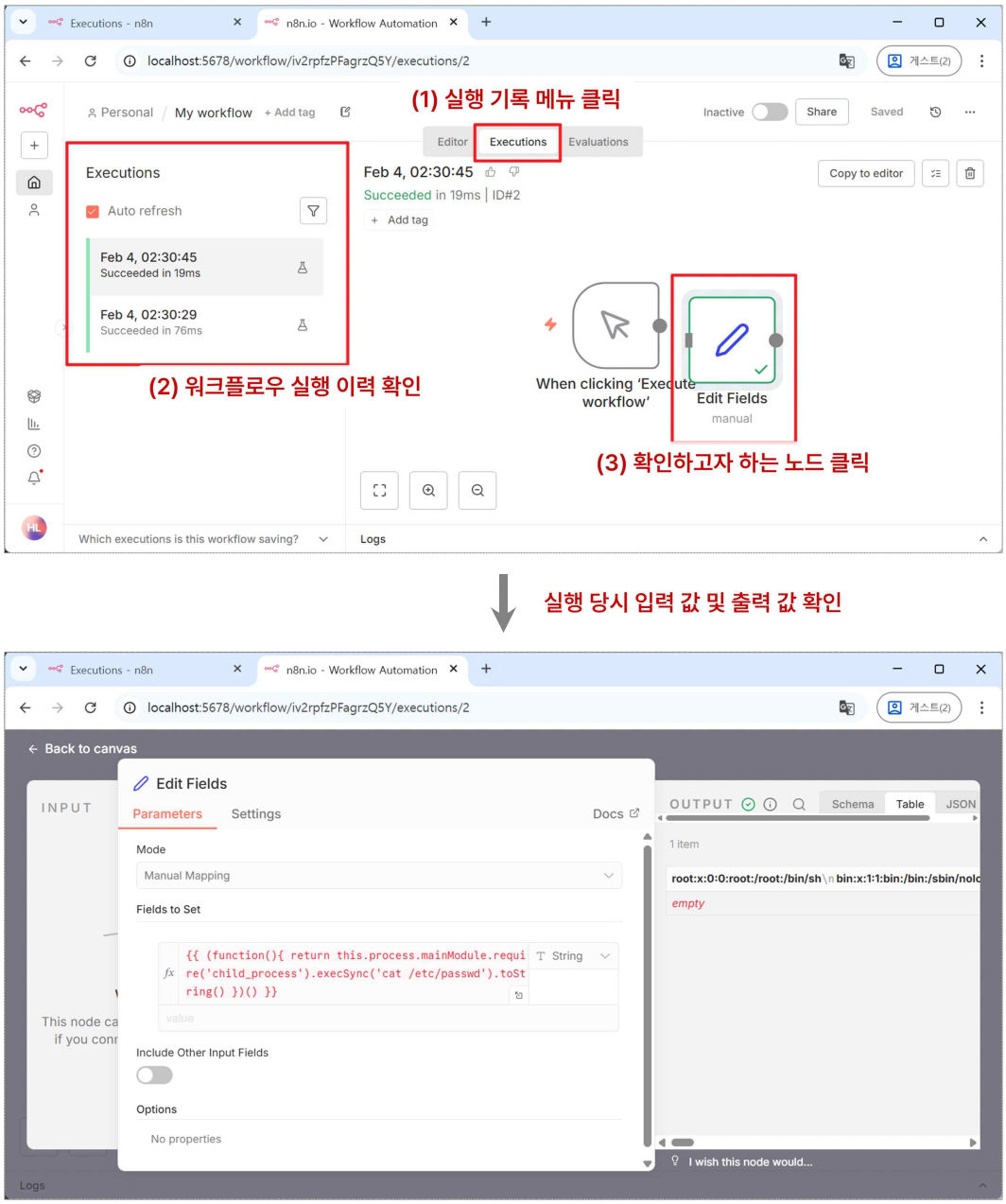

## 4. 악성 워크플로우 실행 이력 확인 방안

본 절에서는 CVE-2025-68613 취약점이 실제로 악용되었는지를 사후적으로 확인하기 위해 n8n에서 생성되는 로그와 워크플로우 실행 이력을 활용해 악성 표현식 실행 흔적을 확인하는 방법을 소개한다. 해당 취약점은 정상적인 워크플로우 실행 흐름 내에서 발생하므로 n8n에서 기록하는 자체 로그 분석이 중요한 단서가 될 수 있다.

### (1) 웹 인터페이스 내 워크플로우 실행 로그 확인

n8n은 워크플로우 실행 시 각 노드의 실행 결과와 오류 정보를 내부 실행 로그로 기록한다. 사용자는 n8n 웹 인터페이스의 실행기록(Executions) 메뉴를 통해 과거 워크플로우 실행 이력을 확인할 수 있으며, 각 실행 건에 대해 노드별 입력 값과 출력 값을 조회할 수 있다.

악성 표현식이 실행된 경우, Set 노드와 같이 표현식을 사용하는 노드의 출력 값에 운영체제 명령 실행 결과가 그대로 포함될 수 있다. 예를 들어, `/etc/passwd` 파일 내용, 시스템 사용자 정보(`id` 명령 결과) 등이 출력 데이터로 확인될 경우, 표현식이 데이터 처리 목적을 벗어나 서버 명령을 실행했을 가능성을 의심할 수 있다. 한편 출력 결과가 명확히 드러나지 않더라도 표현식 자체에 외부 리소스 접근이나 파일 다운로드와 같은 악성 스크립트가 포함된 경우도 존재한다. 예를 들어 `curl`, `wget` 등을 호출해 외부 서버로부터 파일을 다운로드하거나 추가 스크립트를 로드 및 실행하는 표현식은 실행 결과가 노출되지 않더라도 이후 악성 행위로 이어질 가능성이 높다. 따라서 단순히 출력 값만을 기준으로 침해 여부를 판단하기보다는 해당 표현식이 어떤 스크립트로 구성되어 있는지 자체 내용을 함께 검토하는 것이 중요하다.




▲ 실행 기록(Executions) 메뉴를 통한 실행 이력 확인

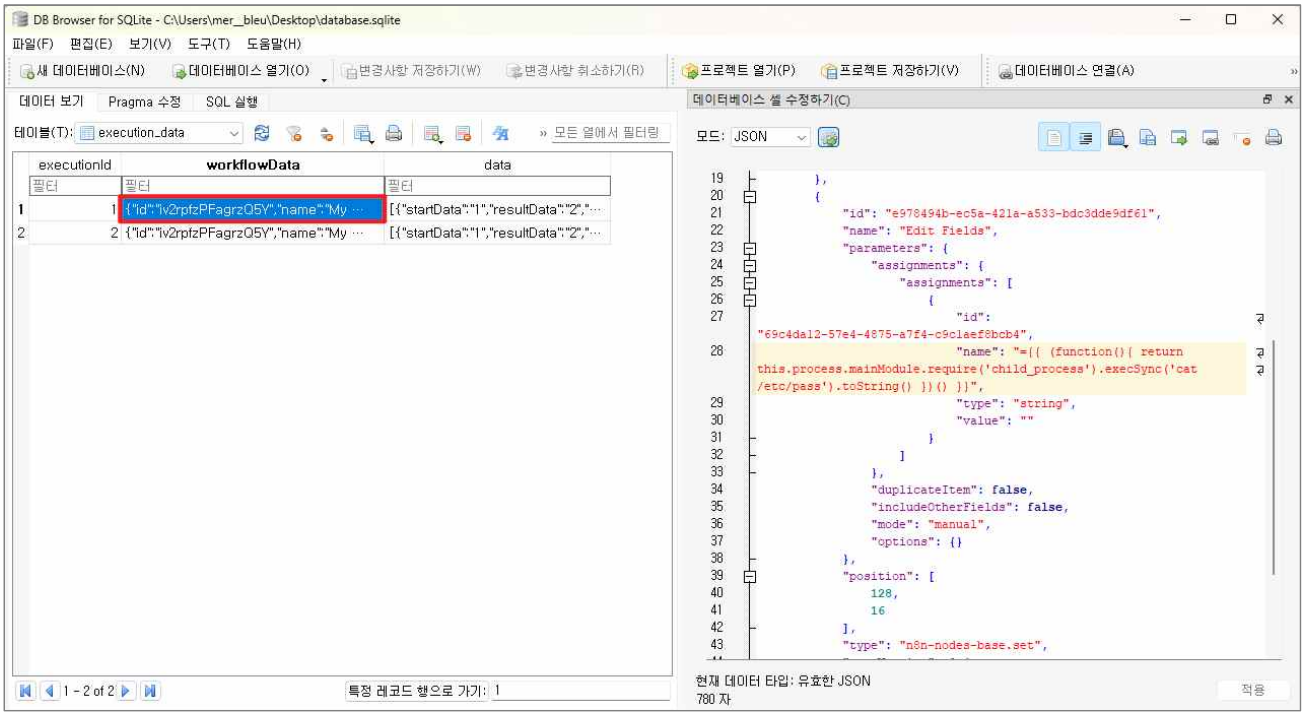
### (2) n8n 내부 데이터베이스 기반 실행 이력 분석

n8n은 워크플로우 구성 정보와 워크플로우 실행 이력, 노드별 실행 데이터 등을 내부 데이터베이스에 구조화 된 형태로 저장한다. 따라서 웹 인터페이스에서 확인 가능한 실행 이력이 되었더라도 해당 데이터베이스를 분석하면 악성 표현식의 실행 흔적을 확인할 수 있다.

n8n은 기본적으로 SQLite를 사용하며 Docker 기반 기본 구성에서는 n8n 사용자 디렉터리인 /home/node/.n8n 경로에 데이터베이스 파일이 존재한다.

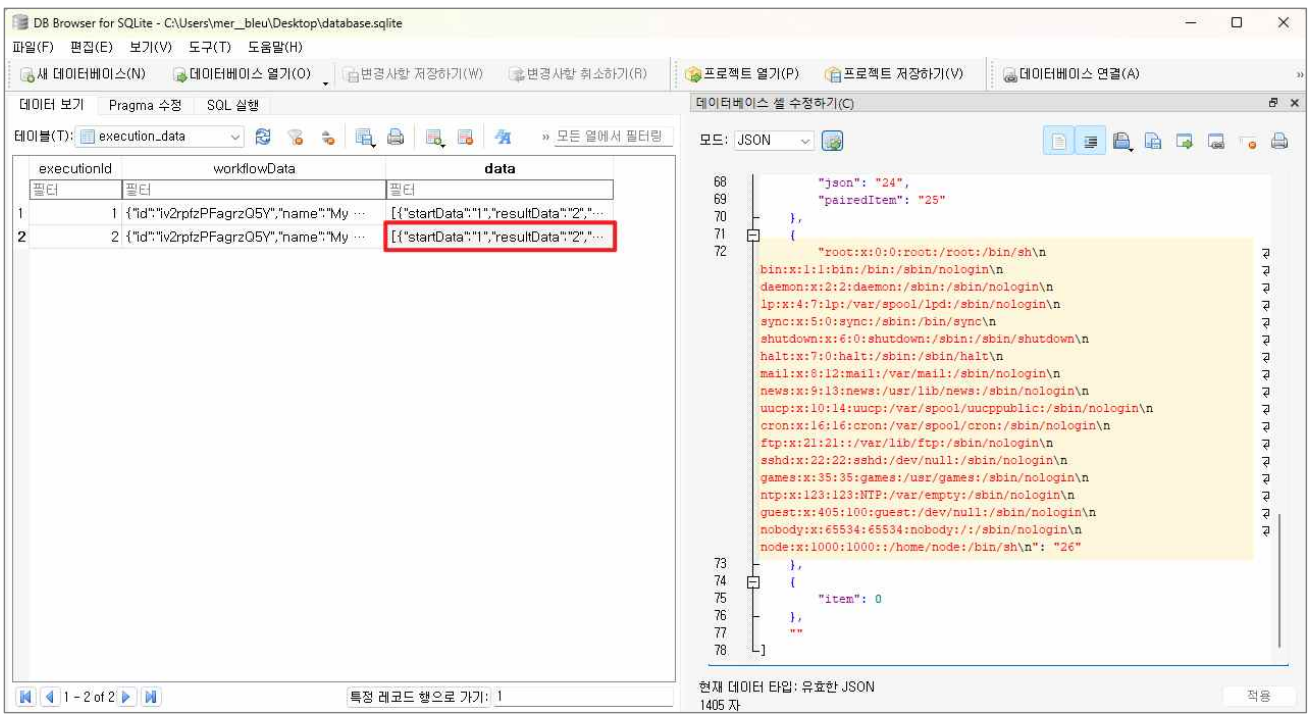
```
/home/node/.n8n/database.sqlite
```

해당 데이터베이스의 '`execution_data`' 테이블에는 실행 당시 워크플로우 스냅샷과 입력 및 출력 데이터가 JSON 형태로 기록된다. '`workflowData`' 열에는 워크플로우 실행 당시의 입력 데이터가 JSON 형태로 저장되어 있어 Set 노드에 입력했던 악성 스크립트 표현식을 확인할 수 있다.




▲ database.sqlite 데이터베이스 내 execution_data 테이블 데이터 (1)

또한, '`data`' 열에는 워크플로우 실행 당시 출력 데이터가 JSON 형태로 담겨 있어 Set 노드에 입력했던 악성 스크립트 표현식이 실행됐던 결과를 확인할 수 있다.




▲ database.sqlite 데이터베이스 내 execution_data 테이블 데이터 (2)

다음 표는 n8n 공식 문서[1]의 데이터베이스 구조 설명을 바탕으로 n8n 침해 분석 과정에서 활용 가치가 높은 주요 테이블과 해당 테이블에 저장되는 데이터를 정리한 것이다.

| 테이블명 | 저장되는 데이터 (요약) |
|---|---|
| workflow_entity | ▪ 인스턴스에 저장된 워크플로우 정의 (노드 구성, 설정, 활성화 상태 등) |
| workflow_history | ▪ 워크플로우의 과거 버전 (변경 이력) |
| execution_entity | ▪ 저장된 워크플로우의 실행 메타데이터 정보 (상태, 시작/종료일시 등) |
| execution_data | ▪ 실행 당시 워크플로우 스냅샷 및 노드 실행 데이터 (입력 및 출력 데이터 포함) |
| credentials_entity | ▪ 외부 연동에 사용하는 자격 증명 정보 |
| shared_workflow | ▪ 워크플로우별 사용자 공유 정보 |
| user | ▪ 사용자 데이터 |
| installed_nodes | ▪ 설치된 커뮤니티 노드 |
| installed_packages | ▪ 설치된 커뮤니티 노드 npm 패키지 정보 |

1) https://docs.n8n.io/hosting/architecture/database-structure/#installed_packages

# [ 04 대응방안 ]

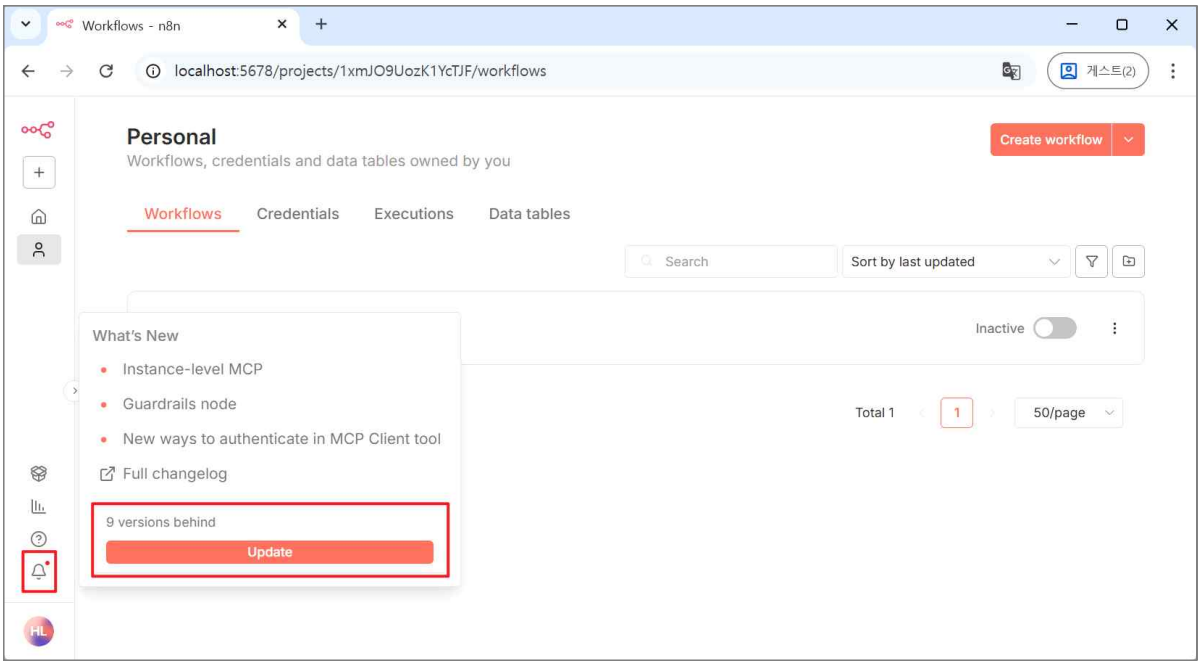
## 1. 패치 및 업그레이드

n8n에서 확인된 CVE-2025-68613 취약점은 표현식(Expression) 실행 과정에서 의도된 실행 컨텍스트를 벗어날 수 있는 구조적 문제가 원인으로, 공격자가 이를 악용할 경우 임의 코드 실행으로 이어질 수 있다. 이에 대해 n8n 개발사는 표현식 실행 범위를 제한하고 보안 검증을 강화한 패치를 배포했다.

해당 취약점을 근본적으로 해결하기 위해서는 보안 패치가 적용된 버전으로 업데이트가 필수적이며, n8n을 이용하고 있는 사용자는 다음 버전 이상으로 즉각적인 업데이트가 필요하다. 특히 외부 네트워에 노출된 환경이나 다수 사용자가 워크플로우를 생성하고 편집하는 환경에서는 우선적으로 적용할 필요가 있다.

- 1.120.4 버전
- 1.121.1 버전
- 1.122.0 이상 버전

다음은 n8n 웹 인터페이스 내에서 업데이트를 적용할 수 있는 메뉴이다.

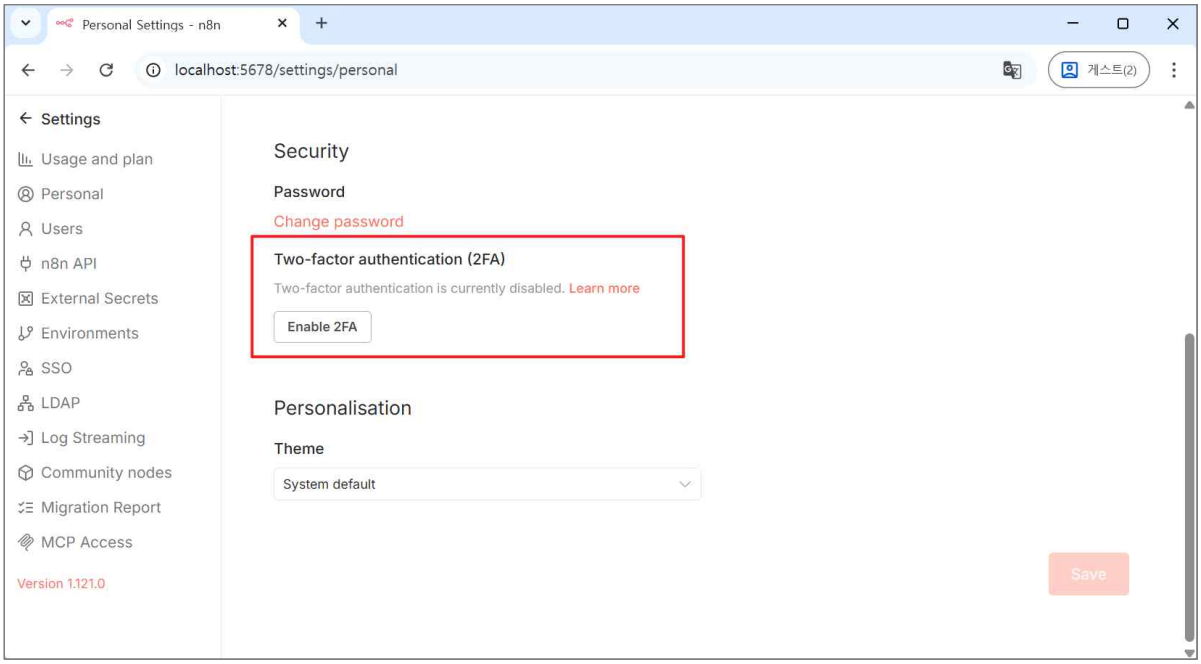



▲ n8n 업데이트 메뉴

## 2. 접근 통제 및 권한 관리 강화

n8n 환경에서 워크플로우 생성 및 편집 권한은 시스템 동작 및 데이터 처리 흐름에 직접적인 영향을 미치므로 최소 권한 원칙에 따라 엄격하게 관리되어야 한다. 워크플로우 생성 및 수정 권한은 소수의 신뢰된 사용자에게만 부여하고 불필요한 관리자 권한 할당은 지양해야 한다.

이를 위해 사용자 계정별 역할을 명확하게 구분하고 정기적인 권한 검토 및 감사 정책을 통해 권한 오·남용 여부를 점검할 필요가 있다. 또한 강력한 비밀번호 정책을 적용하고 2단계 인증(2FA) 또는 기업 환경의 경우 SSO 연계를 통해 계정 탈취 위험을 최소화하는 것이 바람직하다. 특히 외부 협력 인력이나 임시 계정의 경우, 사용 종료 시 즉시 권한을 회수하는 운영 정책이 마련되어야 한다.




▲ n8n 내 사용자 2FA 인증 활성화 기능

## 3. 임시 대응방안

즉각적인 업데이트 적용이 어려운 경우, 취약점 악용 가능성을 낮추기 위한 임시적인 대응방안을 적용할 수 있다. 다만 이러한 조치는 근본적인 해결책이 아니며 패치 적용 전까지의 단기적인 대응 수단으로만 활용되어야 한다.

우선 n8n 서비스는 root 또는 관리자 권한이 아닌 최소 권한을 가진 계정으로 실행되도록 구성해야 하며, 외부 네트워크와의 연결은 업무상 필요한 범위로만 제한해야 한다. 관리 인터페이스 접근은 신뢰된 IP 주소 또는 VPN 환경에서만 허용하도록 설정해 외부 공격 노출을 최소화해야 한다.

추가적으로 Docker 컨테이너 또는 가상머신(VM) 환경에서 운영함으로써 침해 발생 시 피해 범위를 제한할 수 있도록 격리 구조를 적용하는 것도 고려할 수 있다.

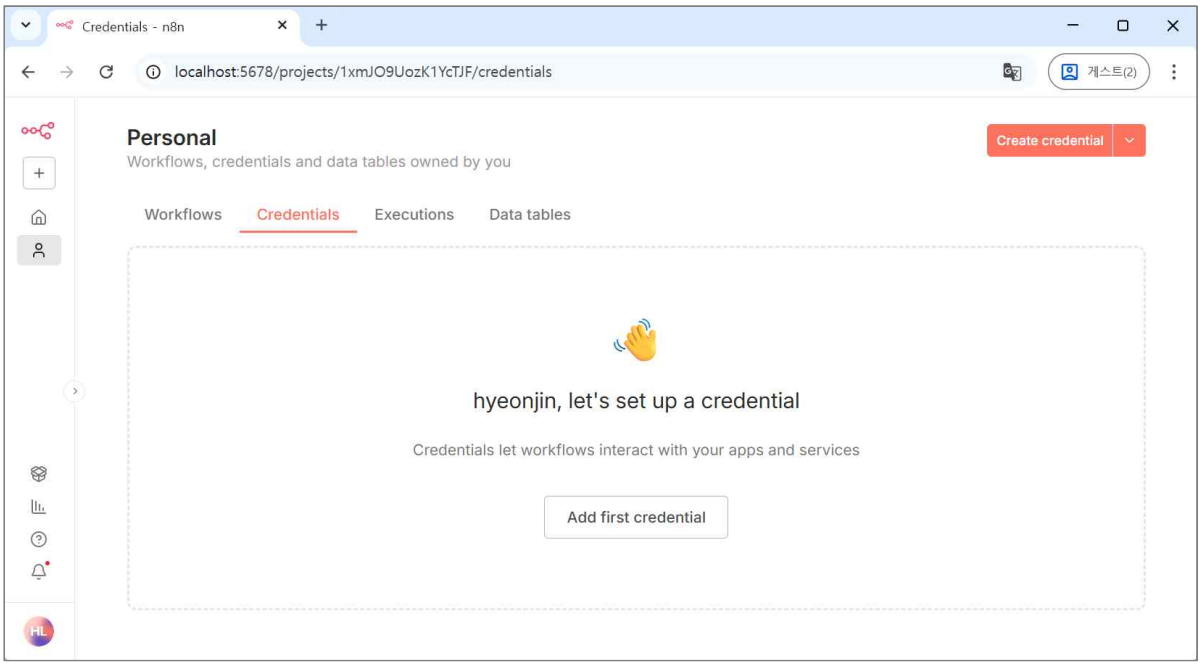
## 4. 워크플로우 및 표현식 관리 정책

운영 중인 워크플로우에 포함된 표현식은 실행 시 다양한 시스템 자원과 데이터에 접근할 수 있으므로, 정기적인 검토와 관리가 필수적이다. 특히 사용자 정의 표현식이 포함된 워크플로우에 대해서는 코드 수준의 검토 절차를 마련해 의도하지 않은 동작이나 보안상 위험 요소가 포함되어 있는지 확인해야 한다.

중요 업무에 사용되는 핵심 워크플로우는 명명 규칙을 통해 식별 가능하도록 관리하고, 변경 시 승인 절차 및 변경 이력 관리가 이루어지도록 운영 정책을 강화할 필요가 있다. 또한 데이터 손실이나 변조에 대비해 워크플로우 정의 파일에 대한 정기적인 백업을 수행하고 복구 절차를 사전에 마련해두는 것이 바람직하다.

## 5. 자격 증명 관리

워크플로우 내에 계정 정보나 API 키 등 자격 증명을 직접 하드코딩하는 방식은 지양해야 하며, 민감한 정보는 환경 변수 또는 n8n에서 제공하는 보안 자격 증명 관리 기능을 활용해 분리하여 관리해야 한다.




▲ 자격 증명 관리 기능

또한, n8n이 활용하는 모든 자격 증명에 대해 접근 권한을 제한하고 정기적인 변경 정책을 적용함으로써 노출 위험을 최소화할 필요가 있다. 아울러 자격 증명 접근 이력에 대한 감사 로그를 유지하여 비정상 접근이나 오남용 여부를 사후에 추적할 수 있도록 해야 한다.

# [ 05 결론 ]

CVE-2025-68613은 n8n의 핵심 자동화 요소인 표현식(Expression) 기능에서 발생한 보안 취약점으로, 인증된 사용자에 의해 원격 코드 실행이 가능하다는 점에서 심각한 위협으로 평가된다. 본 취약점은 단순한 입력 검증 미흡이나 특정 기능상의 오류가 아니라 표현식이 서버 측에서 자바스크립트 코드로 해석·실행되는 구조와 실행 컨텍스트가 적절히 분리되지 않은 설계로 인해 발생하였다.

본 보고서에서 분석한 바와 같이 n8n의 표현식은 자바스크립트 문법을 그대로 지원하며 취약한 버전에서는 `this` 구문을 통해 Node.js 런타임의 전역 객체에 접근할 수 있었다. 이로 인해 표현식이 본래 의도된 데이터 처리 범위를 벗어나 내부 API 호출이나 운영체제 명령 실행으로까지 확장될 수 있는 공격 경로가 형성되었다. 이러한 구조적 문제는 자동화 플랫폼이 가지는 높은 권한 집중성과 결합되면서 단일 워크플로우를 통한 서버 장악 및 추가 침해로 이어질 수 있는 현실적인 위협을 만든다.

특히 본 취약점은 관리자 권한 없이도 악용이 가능하며 워크플로우 생성 또는 편집 권한을 가진 일반 사용자 계정만으로도 공격이 가능하다는 점에서 위험성이 크다. 이로 인해 내부자 위협이나 계정 탈취 사고가 발생할 경우 피해 범위가 급격히 확대될 수 있다. 이는 자동화 플랫폼을 단순한 업무 편의 도구가 아닌 조직의 핵심 인프라 구성 요소로 인식하고 보다 엄격한 보안 관리가 필요함을 시사한다.

본 보고서는 취약점의 동작 원리와 공격 시나리오 분석에 더해 n8n의 실행 로그와 내부 데이터베이스 구조를 활용한 침해 여부 분석 방안을 제시하였다. 이를 통해 단순히 취약점 패치 여부를 확인하는 수준을 넘어 실제 침해 발생 여부를 검증하고 과거 악성 워크플로우 실행 흔적을 식별할 수 있는 접근 방법을 제안하고자 했다.

결론적으로 n8n을 포함한 자동화 플랫폼을 운영하는 조직은 기능적 편의성과 생산성뿐만 아니라 서버 측 코드 실행 경로와 권한 구조에 대한 보안 검증을 필수적으로 고려해야 한다. CVE-2025-68613 사례는 자동화 플랫폼의 설계 단계에서 보안적 고려가 부족할 경우 어떤 위협으로 이어질 수 있는지를 보여주는 대표적인 사례이며, 향후 유사한 취약점의 재발을 방지하기 위해서는 지속적인 패치 관리, 권한 통제, 워크플로우 검토 체계 구축이 병행되어야 할 것이다.