resetPassword Authentication Bypass Vulnerability

High **igor-magun-wd** published **GHSA-f6hc-c5jr-878p** last week

Package

flowise (npm)

Affected versions	Patched versions
<= 3.0.13	3.1.0

Description

poc file: https://trendmicro-my.sharepoint.com/:u:/p/kholoud_altookhy/IQD47asMXogJQZHvNkYE49k7AWaW-VFnUMacZc1WLkqj16k?e=rjSaQc
ZDI-CAN-28762: Flowise AccountService resetPassword Authentication Bypass Vulnerability

-- CVSS ----------

8.1: AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

-- ABSTRACT ----------

Trend Micro's Zero Day Initiative has identified a vulnerability affecting the following products:
Flowise - Flowise

-- VULNERABILITY DETAILS ----------

- Version tested: 3.0.12
- Installer file: hxxps://github.com/FlowiseAI/Flowise
- Platform tested: NA

Analysis

This vulnerability allows remote attackers to bypass authentication on affected installations of FlowiseAI Flowise. Authentication is not required to exploit this vulnerability.

The specific flaw exists within the resetPassword method of the AccountService class. The issue results from improper implementation of the authentication mechanism. An attacker can leverage this vulnerability to change user's passwords and bypass authentication on the system.

Product information

FlowiseAI Flowise version 3.0.12 (hxxps://github.com/FlowiseAI/Flowise)

Setup Instructions

```
npm install flowise@3.0.12
npx flowise start
```

Root Cause Analysis

FlowiseAI Flowise is an open source low-code tool for developers to build customized large language model (LLM) applications and AI agents. It supports integration with various LLMs, data sources, and tools in order to facilitate rapid development and deployment of AI solutions. Flowise offers a web interface with a drag-and-drop editor, as well as an API, through an Express web server accessible over HTTP on port 3000/TCP.

Flowise allows users to reset forgotten passwords using a token emailed to the email address associated with their account. A token is sent to the user's email when a request is made to the "/api/v1/account/forgot-password" endpoint. Users will submit this token along with their new password to the "/api/v1/account/reset-password" endpoint, and if it is submitted within sufficient time (15 minutes by default, or the value of the PASSWORD_RESET_TOKEN_EXPIRY_IN_MINUTES environment variable) the user will be able to change their password.

The resetPassword() method of the AccountService class is responsible for handling such requests. This method will first retrieve the account information of the user based on their email address, which includes the value of the reset token. The method will then check if the reset token provided matches the one stored in the user's account, and that the token hasn't expired, before changing that users password.

However, there is no check performed to ensure that a password reset token has actually been generated for a user account. By default the value of the reset token stored in a users account is null, or an empty string if they've reset their password before. An attacker with knowledge of the user's email address can submit a request to the "/api/v1/account/reset-password" endpoint containing a null or empty string reset token value and reset that user's password to a value of their choosing. The null or empty string reset token value will allow the attacker to pass the reset token check, and they only have to worry about passing the expiry time check. By default the expiry time stored in the account of a user that has never generated a reset token before is equal to the time of their accounts creation plus 15 minutes (or the value of the PASSWORD_RESET_TOKEN_EXPIRY_IN_MINUTES environment variable).

This means that an attacker with knowledge of a recently created user account's email can change the user's password and use the changed password to bypass authentication.

comments documenting the issue have been added to the following code snippet. Added comments are prepended with "!!!".
From packages/server/src/enterprise/services/account.service.ts

```
    public async resetPassword(data: AccountDTO) {
        data = this.initializeAccountDTO(data)
        const queryRunner = this.dataSource.createQueryRunner()
        await queryRunner.connect()
        try {
            const user = await this.userService.readUserByEmail(data.user.email, queryRu
            if (!user) throw new InternalFlowiseError(StatusCodes.NOT_FOUND, UserErrorMe
            //!!!user.tempToken is null or empty string, unless a user has requested a r
            if (user.tempToken !== data.user.tempToken)  //!!! check if the stored token
                throw new InternalFlowiseError(StatusCodes.BAD_REQUEST, UserErrorMessage

            const tokenExpiry = user.tokenExpiry
            const now = moment()
            const expiryInMins = process.env.PASSWORD_RESET_TOKEN_EXPIRY_IN_MINUTES
                ? parseInt(process.env.PASSWORD_RESET_TOKEN_EXPIRY_IN_MINUTES)
                : 15
            const diff = now.diff(tokenExpiry, 'minutes') //!!! check if token is expire
            if (Math.abs(diff) > expiryInMins) throw new InternalFlowiseError(StatusCode

            // all checks are done, now update the user password, don't forget to hash i
            // leave the user status and other details as is
            //!!! hash and update the user's password since checks passed
            const salt = bcrypt.genSaltSync(parseInt(process.env.PASSWORD_SALT_HASH_ROUN
            // @ts-ignore
            const hash = bcrypt.hashSync(data.user.password, salt)
            data.user = user
            data.user.credential = hash
            data.user.tempToken = ''  //!!! stored Token value is set to empty string wh
            data.user.tokenExpiry = undefined
            data.user.status = UserStatus.ACTIVE

            await queryRunner.startTransaction()
            data.user = await this.userService.saveUser(data.user, queryRunner)  //!!! s
            await queryRunner.commitTransaction()
```

```
            // Invalidate all sessions for this user after password reset
            await destroyAllSessionsForUser(user.id as string)
        } catch (error) {
            await queryRunner.rollbackTransaction()
            throw error
        } finally {
            await queryRunner.release()
        }

        return sanitizeUser(data.user)
    }
```

Proof of Concept

A proof of concept for this vulnerability is provided in ./poc.py. It expects the following syntax:

```
python3 poc.py --user <USER> --host <HOST> [--port <PORT>]
```

Where USER is the email address of a user on the server, HOST is the ip address of the vulnerable server, and PORT is the port the vulnerable server is listening on (default: 3000). Options inclosed in square brackets are optional.

In order for this proof of concept to be successful, the user specified as the USER argument must have created their account within the last 15 minutes (or within PASSWORD_RESET_TOKEN_EXPIRY_IN_MINUTES minutes)

By default, the poc will first send a POST request to the "/api/v1/account/reset-password" endpoint of the target server containing a JSON body with a null reset token and the password "TMSR1234!" for the user specified by the USER argument. If the request doesn't successfully change the user's password, the same request will be sent again with the reset token value set to the empty string. Upon successful exploitation, the user's password will be changed to "TMSR1234!".

The provided proof of concept was tested using FlowiseAI Flowise version 3.0.12 runing on a Ubuntu 24.04 VM.

Credits

Nicholas Zubrisky (@NZubrisky) of TrendAI Research

-- CREDIT ----------
This vulnerability was discovered by:
Nicholas Zubrisky (@NZubrisky) of TrendAI Research of Trend Micro

-- FURTHER DETAILS ----------

Supporting files:

If supporting files were contained with this report they are provided within a password protected ZIP file. The password is the ZDI candidate number in the form: ZDI-CAN-XXXX where XXXX is the ID number.

Please confirm receipt of this report. We expect all vendors to remediate ZDI vulnerabilities within 120 days of the reported date. If you are ready to release a patch at any point leading up to the deadline, please coordinate with us so that we may release our advisory detailing the issue. If the 120-day deadline is reached and no patch has been made available we will release a limited public advisory with our own mitigations, so that the public can protect themselves in the absence of a patch. Please keep us updated regarding the status of this issue and feel free to contact us at any time:

Zero Day Initiative
zdi-disclosures@trendmicro.com

The PGP key used for all ZDI vendor communications is available from:

http://www.zerodayinitiative.com/documents/disclosures-pgp-key.asc

-- INFORMATION ABOUT THE ZDI ----------
Established by TippingPoint and acquired by Trend Micro, the Zero Day Initiative (ZDI) neither re-sells vulnerability details nor exploit code. Instead, upon notifying the affected product vendor, the ZDI provides its Trend Micro TippingPoint customers with zero day protection through its intrusion prevention technology. Explicit details regarding the specifics of the vulnerability are not exposed to any parties until an official vendor patch is publicly available.

Please contact us for further details or refer to:

http://www.zerodayinitiative.com

-- DISCLOSURE POLICY ----------

Our vulnerability disclosure policy is available online at:

http://www.zerodayinitiative.com/advisories/disclosure_policy/

Severity

High 7.7 / 10

CVSS v4 base metrics

Exploitability Metrics

Attack Vector	Network
Attack Complexity	High
Attack Requirements	Present
Privileges Required	None
User interaction	Passive

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:H/AT:P/PR:N/UI:P/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

CVE ID

CVE-2026-41276

Weaknesses

No CWEs

Credits

zdi-disclosures Reporter