Vulnerability call chain

1.1 Summary

SpringBlade's `POST /user/user-auth-info` endpoint accepts a client-supplied `userId` and persists it with an OAuth subject identified by `uuid` and `source`. An attacker who controls an OAuth identity can bind that identity to an arbitrary local account, including an administrator account. A later OAuth login using the same identity resolves to the victim account and receives its authentication context.

- Attack precondition: the attacker can reach `/user/user-auth-info` and control an OAuth `uuid/source` pair. A low-privileged JWT may be required depending on deployment exposure.
- Affected endpoint: `POST /user/user-auth-info`
- Affected authorization properties: `UserOauth.userId`, `UserOauth.tenantId`, `UserOauth.uuid`, and `UserOauth.source`
- Security impact: persistent OAuth account binding hijack, account takeover, privilege escalation, and possible disclosure of serialized password hashes

The issue was dynamically verified against an isolated SpringBlade `v4.10.0` deployment containing `blade-auth`, `blade-system`, the SpringBlade Gateway, Nacos, MySQL, and Redis.

1.2 Severity

The authenticated-attacker case is High severity, CVSS 3.1 `8.8`:

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

If the endpoint is anonymously reachable through gateway routing or direct service exposure, the impact is Critical, CVSS 3.1 `9.8`:

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

Production deployments should verify gateway, service-discovery, ingress, and WAF exposure.

1.3 Exploit path

1. The attacker submits `UserOauth` with an attacker-controlled `uuid/source` pair and `userId` set to a target local user.
2. `UserServiceImpl.userInfo(UserOauth)` searches `blade_user_oauth` by `uuid` and `source`.
3. If no row exists, the service saves the complete request object, including the attacker-selected `userId`.
4. If a row exists, the service loads the local user referenced by the stored `userId`.
5. `SocialTokenGranter` uses the returned `UserInfo` to complete the OAuth login and issue an authentication token.

1.4 Proof of concept

In the test environment, User B submitted:

```
POST /blade-system/user/user-auth-info
Blade-Auth: bearer <B-low-privileged-JWT>
Content-Type: application/json

{
  "tenantId": "000000",
  "uuid": "gateway-b-oauth-3",
  "source": "github",
  "userId": "1123598821738675201",
  "username": "gateway-b"
}
```

The initial response was a guest user, but the application persisted:

```
gateway-b-oauth-3 -> 1123598821738675201 (admin)
```

A subsequent request with the same `uuid` and `source` returned:

```
user.id: 1123598821738675201
account: admin
```

```
roles: [administrator]
```

The reverse direction was also reproduced: `gateway-a-oauth-1` was associated with user ID `1123598821738675202`, and subsequent resolution returned account `userb`.

1.5 Key code evidence

1. `IUserClient.java` exposes the endpoint as a REST mapping:

Evidence location: IUserClient.java#L69-L70

```
@PostMapping(API_PREFIX + "/user-auth-info")
R<UserInfo> userAuthInfo(@RequestBody UserOauth userOauth);
```

1. `UserClient.java` forwards the request directly to the service:

Evidence location: UserClient.java#L53-L56

```
@PostMapping(API_PREFIX + "/user-auth-info")
public R<UserInfo> userAuthInfo(UserOauth userOauth) {
    return R.data(service.userInfo(userOauth));
}
```

1. `UserServiceImpl.java` looks up only `uuid` and `source`, then trusts the stored `userId`:

Evidence location: UserServiceImpl.java#L118-L129

```
@Transactional(rollbackFor = Exception.class)
public UserInfo userInfo(UserOauth userOauth) {
    UserOauth uo = userOauthService.getOne(
        Wrappers.<UserOauth>query().lambda()
            .eq(UserOauth::getUuid, userOauth.getUuid())
            .eq(UserOauth::getSource, userOauth.getSource()));
    if (Func.isNotEmpty(uo) && Func.isNotEmpty(uo.getUserId())) {
        userInfo = this.userInfo(uo.getUserId());
    } else if (Func.isEmpty(uo)) {
```

```
        userOauthService.save(userOauth);
    }
```

1. `SocialTokenGranter.java` constructs the OAuth binding and calls the endpoint during login:

Evidence location: SocialTokenGranter.java#L81-L87

```
UserOauth userOauth = Objects.requireNonNull(BeanUtil.copyProperties(authUser, U
userOauth.setSource(authUser.getSource());
userOauth.setTenantId(tenantId);
userOauth.setUuid(authUser.getUuid());
R<UserInfo> result = userClient.userAuthInfo(userOauth);
```

2. Impact

An attacker who controls any OAuth identity can associate it with an arbitrary local account by providing a chosen `userId`. Successful exploitation may lead to:

- Full account takeover
- Privilege escalation to administrator accounts
- Persistent malicious associations in `blade_user_oauth`
- Future OAuth logins authenticating as the victim
- Disclosure of stored password hashes when `UserInfo.user` serializes the password field

The attacker does not need the victim's OAuth account, OAuth credentials, or password. The attacker needs only control of an OAuth identity and knowledge or enumeration of a target local user ID.

3. Root cause analysis

Root Cause 1: Publicly reachable Feign endpoint.

Feign usage alone does not restrict direct HTTP access. The interface is exposed through a normal REST controller, so external callers may invoke it like any other API when routing permits.

Root Cause 2: Unsafe trust of client-supplied `userId`.

The implementation persists the complete client-supplied `UserOauth` object when no binding exists and later loads the account referenced by its `userId`, without checking caller ownership, OAuth ownership, tenant boundaries, or authorization scope.

Root Cause 3: Missing service-to-service and account-binding controls.

The endpoint does not require trusted service authentication, does not derive the target account from the current authenticated user, and does not enforce explicit user consent for attaching an OAuth identity.

4. Recommended fix

1. Make the endpoint internal-only and protect it with service-to-service authentication such as mTLS, workload identity, signed requests, and network isolation.
2. Remove client control over `id`, `userId`, and `tenantId`. OAuth lookup requests should contain only provider-verified `uuid`, `source`, and server-derived tenant context.
3. Implement account linking as a separate authorized workflow requiring an authenticated local session, verified OAuth ownership, tenant validation, and explicit user consent.

The security invariant for account linking must be:

```
requestedUserId == authenticatedCurrentUserId
```

5. Verification after fix

- Unauthorized callers receive HTTP 403 or an equivalent rejection.
- Out-of-scope target identifiers are rejected before database writes or sensitive reads.
- Tenant and ownership boundaries are enforced server-side.
- Direct HTTP requests to the internal endpoint are rejected even when front-end controls are hidden.

Published reference: https://aibot88.github.io/CVERequest/SpringBlade/issue2.html