OAuth account confusion via shared mutable state on singleton service instances

High **steveiliop56** published **GHSA-9q5m-jfc4-wc92** 5 days ago

Package

github.com/steveiliop56/tinyauth (Go)

Affected versions	Patched versions
<5.0.5	5.0.5

Description

Summary

All three OAuth service implementations (`GenericOAuthService`, `GithubOAuthService`, `GoogleOAuthService`) store PKCE verifiers and access tokens as mutable struct fields on singleton instances shared across all concurrent requests. When two users initiate OAuth login for the same provider concurrently, a race condition between `VerifyCode()` and `Userinfo()` causes one user to receive a session with the other user's identity.

Details

The `OAuthBrokerService.GetService()` returns a single shared instance per provider for every request. The OAuth flow stores intermediate state as struct fields on this singleton:

Token storage — `generic_oauth_service.go` line 96:

```
generic.token = token  // Shared mutable field on singleton
```

Verifier storage — `generic_oauth_service.go` line 81:

```
generic.verifier = verifier  // Shared mutable field on singleton
```

In the callback handler `oauth_controller.go` lines 136–143, the code calls:

```
err = service.VerifyCode(code)                          // line 136 — stores token on l
// ... race window ...
user, err := controller.broker.GetUser(req.Provider)  // line 143 — reads token from sin
```

Between these two calls, a concurrent request's `VerifyCode()` can overwrite the `token` field, causing `GetUser()` → `Userinfo()` to fetch the **wrong user's** identity claims.

The same pattern exists in all three implementations:

- `github_oauth_service.go` lines 34–39, 77, 86–99
- `google_oauth_service.go` lines 22–27, 65, 73–87

PoC

Race scenario (two concurrent OAuth callbacks):

1. User A and User B both click "Login with GitHub" on the same tinyauth instance
2. Both are redirected to GitHub, authorize, and GitHub redirects both back with authorization codes
3. Both callbacks arrive at tinyauth nearly simultaneously:

```
Timeline:
  t0: Request A → service.VerifyCode(codeA) → singleton.token = tokenA
  t1: Request B → service.VerifyCode(codeB) → singleton.token = tokenB  (overwrites
tokenA)
  t2: Request A → broker.GetUser("github")  → Userinfo() reads singleton.token =
tokenB
  t3: Request A receives User B's identity (email, name, groups)
```

User A now has a tinyauth session with User B's email, gaining access to all resources User B is authorized for via tinyauth's ACL.

PKCE verifier DoS variant: Even with PKCE, concurrent `oauthURLHandler` calls overwrite the `verifier` field, causing `VerifyCode()` to send the wrong verifier to the OAuth provider, which rejects the exchange.

Static verification: Run Go's race detector on a test that calls `VerifyCode` and `Userinfo` concurrently on the same service instance — the `-race` flag will flag data races on the `token` and `verifier` fields.

Go race detector confirmation: Running a concurrent test with `go test -race` on the singleton service detects **4 data races** on the `token` and `verifier` fields. Without the race detector, measured token overwrite rate is 99.9% (9,985/10,000 iterations).

Test environment: tinyauth v5.0.4, commit `592b7ded`, Go race detector + source code analysis

Impact

An attacker who times their OAuth callback to race with a victim's callback can obtain a tinyauth session with the victim's identity. This grants unauthorized access to all resources the victim is permitted to access through tinyauth's ACL system. The probability of collision increases with concurrent OAuth traffic.

The PKCE verifier overwrite additionally causes a denial-of-service: concurrent OAuth logins for the same provider reliably fail.

Suggested Fix

Pass verifier and token through method parameters or return values instead of storing them on the singleton:

```
func (generic *GenericOAuthService) VerifyCode(code string, verifier string) (*oau o
    return generic.config.Exchange(generic.context, code, oauth2.VerifierOption(ve r
}

func (generic *GenericOAuthService) Userinfo(token *oauth2.Token) (config.Claims, error)
    client := generic.config.Client(generic.context, token)
    // ...
}
```

Store the PKCE verifier in the session/cookie associated with the OAuth `state` parameter, not on the service struct.

Severity

High 7.7 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-33544

Weaknesses

▶ CWE-362

Credits

 **kq5y** Reporter