

New issue

[Jump to bottom](#)

# Race condition can make MaxCreationLimit useless #5926

 Open shouc opened this issue on Feb 19, 2020 · 7 comments

Labels

 bug

shouc commented on Feb 19, 2020

## Description

Users could potentially create more repos than specified in MaxCreationLimit as NumRepos field is not updated in a race-condition-safe cavalier (i.e. row is not locked). Such logic error could be fatal in some specific settings.

## Reason

internal/db/repo.go:1108

```
func CreateRepository(doer, owner *User, opts CreateRepoOptions) (_ *Repository, err error) {
    // check first
    if !owner.CanCreateRepo() {
        return nil, errors.ReachLimitOfRepo{owner.RepoCreationNum()}
    }
    // some time-consuming operations
    ...
    // update numrepo
    if err = createRepository(sess, doer, owner, repo); err != nil {
        return nil, err
    }
    ...
}
```

## PoC

Execute following script in the console of the user:

```
[0,1,2,3,4,5,6,7,8,9].forEach((v)=>{$.post("/repo/create", "_csrf=[YOUR CSRF TOKEN]&user_id=[YOUR ID]&repo_name=ccc" + v +"&description=&gitignores=&license=&readme=Default"))})
```





The resultant NumRepos is less than 10 (it is 2~4 in my settings) though 10 repos are created.

## Solution

Indeed, some other fields also need locking but are not that crucial to the integrity of the system.

```
type User struct {
    ...
    NumFollowers int // <- add optimistic lock here
    NumFollowing int `xorm:"NOT NULL DEFAULT 0"` // <- add optimistic lock here
    NumStars     int // <- add optimistic lock here
    NumRepos     int // <- add optimistic lock here
    ...
}
```

Or stop using fields in user table to save the value as it could be counted directly by using repository table.

 4  unknown added  good first issue  bug labels on Feb 19, 2020

unknownw commented on Feb 19, 2020

Member

Thank you for a high quality report!

jonatan5524 commented on Jul 17

Interesting issue, I would like to fix this.

jonatan5524 commented on Jul 17

as @shouc suggested using an optimistic lock, there is a plugin solution in [gorm](#), which causes the type to be Version type and every access to the field need is with version.Int64 (the type now is int64). I know you @unknownw don't like to involve more dependencies. what do you think?

unknownw commented on Aug 11

Member

as @shouc suggested using an optimistic lock, there is a plugin solution in [gorm](#), which causes the type to be Version type and every access to the field need is with version.Int64 (the type now is int64). I know you @unknwon don't like to involve more dependencies. what do you think?

Thanks for expressing your interests! I think it is probably an overkill, given Gogs is a single binary monolith application, we can simply hold locks in memory. Something like (not a working solution):

```
type createRepositoryLocker struct {
    lock sync.Mutex
    userIDs map[int64]sync.Mutex
}

func (l *createRepositoryLocker) Lock(userID int64) {
    l.lock.Lock()
    user := l.userIDs[userID]
    l.lock.Unlock()

    user.Lock()
}

func (l *createRepositoryLocker) Unlock(userID int64) {
    l.lock.Lock()
    user := l.userIDs[userID]
    l.lock.Unlock()

    user.Unlock()
}
```

jonatan5524 commented on Aug 12

as @shouc suggested using an optimistic lock, there is a plugin solution in [gorm](#), which causes the type to be Version type and every access to the field need is with version.Int64 (the type now is int64). I know you @unknwon don't like to involve more dependencies. what do you think?

Thanks for expressing your interests! I think it is probably an overkill, given Gogs is a single binary monolith application, we can simply hold locks in memory. Something like (not a working solution):

```
type createRepositoryLocker struct {
    lock sync.Mutex
    userIDs map[int64]sync.Mutex
}

func (l *createRepositoryLocker) Lock(userID int64) {
    l.lock.Lock()
    user := l.userIDs[userID]
    l.lock.Unlock()

    user.Lock()
}

func (l *createRepositoryLocker) Unlock(userID int64) {
    l.lock.Lock()
    user := l.userIDs[userID]
    l.lock.Unlock()

    user.Unlock()
}
```

Does it really necessary to create a struct for every single counter?

Maybe add in the User struct a Mutex for each counter or create a struct "safety counters" that there will be the counters and for each count a Mutex.  
proposal for option 1:

```
// Counters
NumFollowersLock sync.Mutex
NumFollowers int
NumFollowingLock sync.Mutex
NumFollowing int `xorm:"NOT NULL DEFAULT 0" gorm:"not null;default:0"`
NumStarsLock sync.Mutex
NumStars int
NumReposLock sync.Mutex
NumRepos int
```

option 2:

```
// Counters
Counters *SafeCounters

type SafeCounters struct {
    NumFollowersLock sync.Mutex
    NumFollowers int
    NumFollowingLock sync.Mutex
    NumFollowing int `xorm:"NOT NULL DEFAULT 0" gorm:"not null;default:0"`
    NumStarsLock sync.Mutex
    NumStars int
    NumReposLock sync.Mutex
    NumRepos int
}
```

What do you think?

unknwon commented on Aug 12 • edited

Member

@jonatan5524

1. For this issue, we only need to deal with the race condition for `MaxCreationLimit`, other counters are out of scope (whether or not they have race conditions is also a separate discussion.)
2. While we have a major refactoring undergoing for database layer, `repos` seems a perfect place to keep the "locker" (though not all code paths are using this new `repos` at the moment, but I think that's fine, once all code paths are migrated, race condition will be solved).

jonatan5524 commented on Aug 12

@jonatan5524

1. For this issue, we only need to deal with the race condition for `MaxCreationLimit`, other counters are out of scope (whether or not they have race conditions is also a separate discussion :)
2. While we have a major refactoring undergoing for database layer, `repos` seems a perfect place to keep the "locker" (though not all code paths are using this new `repos` at the moment, but I think that's fine, once all code paths are migrated, race condition will be solved).

I don't fully understand where or how to achieve this, maybe I need to get more familiar with the code base so I will search for other open issues, If you have something to recommend I would like that :)

  unknown removed the  good first issue label on Aug 15

Assignees

No one assigned

Labels

 bug

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

3 participants

