

New issue

Jump to bottom

Ensure finalizers don't deallocate GPGME objects while C code is still using them #23

Merged proglottis merged 3 commits into proglottis:master from mtrmac:lifetimes on Jan 15, 2020

Conversation 13 Commits 3 Checks 0 Files changed 2



mtrmac commented on Jan 10, 2020 • edited

Contributor

For some time, we've been struggling with more or less occasional memory corruption, e.g. when running the tests of github.com/containers/image/signature

```
GO111MODULE="on" go test -c ./signature/
for i in $(seq 1 10000); do
  echo ===== $i
  (cd signature; GOTRACEBACK=crash GODEBUG=clobberfree=1,cgocheck=2 ./signature.test) || break
done
```

eventually crashes on most platforms.

Thanks to Ulrich Obergfell uobergfe@redhat.com, we now have a strong candidate for the cause:

As far as Go is concerned, `Data.dh` and `Data.cbc` are just values, so it is OK to implement

```
cgo_call(d.dh)
```

as

```
tmp := d.dh
dropReferenceTo(d)
// d's finalizer can be called at any point from now on, possibly deallocating d.dh
cgo_call(tmp) // possibly using free memory
```

To fix this, explicitly extend the lifetime of `Data`:

```
cgo_call(d.dh)
runtime.KeepAlive(d)
```

becomes

```
tmp := d.dh
cgo_call(tmp) // d.dh is certainly still valid
dropReferenceTo(d)
// d's finalizer can be called from now on, possibly deallocating d.dh
```

and the like.

This PR implements that throughout the codebase; see the individual commit messages for details.

I'm filing this PR for ~early review, I'll run a stress-test to validate the fixes (at least on the code paths relevant to us) over the weekend.

mtrmac added 3 commits 3 years ago

- Ensure Data is not dellocated while C code is still using it ...

d43d199

- Ensure Context is not dellocated while C code is still using it ...

7e8c79d

- Ensure Key is not dellocated while C code is still using it ...

d575e5d

mtrmac commented on Jan 10, 2020

Contributor Author

A more explicit demonstration:

```
package main

import (
    "fmt"
    "math/rand"
    "runtime"
)

type pointerBinding struct {
    managed uint64
}

func newPointerBinding() *pointerBinding {
    ptr := &pointerBinding{}
    runtime.SetFinalizer(ptr, (*pointerBinding).Finalizer)
    ptr.managed = rand.Uint64() // Could come from CGO
    fmt.Printf("Created %p, managed value %v\n", ptr, ptr.managed)
    return ptr
}
```

```

func (ptr *pointerBinding) Finalizer() {
    fmt.Printf("Finalizing %p, managed value %v\n", ptr, ptr.managed)
    // Free ptr.managed via CGO
    ptr.managed = 0
}

func (ptr *pointerBinding) functionBinding() {
    fmt.Printf("Calling function for %p, managed value %v\n", ptr, ptr.managed)
    underlyingFunction(ptr.managed)
    fmt.Printf("Done\n") // No reference to ptr here, or the behavior would change
    // HERE runtime.KeepAlive(ptr)
}

func underlyingFunction(pointer uint64) {
    runtime.GC()
    fmt.Printf("Underlying function, managed value %v\n", pointer)
}

func main() {
    ptr := newPointerBinding()
    ptr.functionBinding()
    runtime.GC()
}

```

As is:

```

$ GOTRACEBACK=crash GODEBUG=clobberfree=1,cgocheck=2 go run ./standalone.go
Created 0xc000014080, managed value 5577006791947779410
Calling function for 0xc000014080, managed value 5577006791947779410
Finalizing 0xc000014080, managed value 5577006791947779410
Underlying function, managed value 5577006791947779410
Done

```

note that "Finalizing" happens *before* "Underlying function" is done with the managed value.

Uncommenting the `runtime.KeepAlive` at "HERE":

```

$ GOTRACEBACK=crash GODEBUG=clobberfree=1,cgocheck=2 go run ./standalone.go
Created 0xc000014080, managed value 5577006791947779410
Calling function for 0xc000014080, managed value 5577006791947779410
Underlying function, managed value 5577006791947779410
Done
Finalizing 0xc000014080, managed value 5577006791947779410

```

works as expected. (Without the GO* variables, the finalizer is actually never run, but that's fine for our purposes as well.)

mtrmac commented on Jan 13, 2020

Contributor Author

I'm filing this PR for ~early review, I'll run a stress-test to validate the fixes (at least on the code paths relevant to us) over the weekend.

I have encountered no crashes during a 3-day test run of the tests mentioned above; before this PR the tests would crash in 1–4 iterations on macOS, and in <2 thousand iterations on Linux; now it survived 1053 iterations on macOS, and 160931 iterations on Linux.

(The different number of iterations is caused by the Linux build of GPG having <https://dev.gnupg.org/T3380> applied.)



proglottis reviewed on Jan 14, 2020

[View changes](#)

proglottis left a comment

Owner

Thanks so much for the PR @mtrmac - that must have been an inordinate amount of effort ❤️

If I knew this would have been the result I definitely would not have used finalisers. In fact I'm wondering if it would be better to remove them, better safe than sorry?

I've left a couple of questions

gpgme.go

Show resolved

gpgme.go

Show resolved

gpgme.go

Show resolved

mtrmac changed the title ~~Ensure finalizers don't deallocate GPGME objects while C code is still using them~~ Ensure finalizers don't deallocate GPGME objects while C code is still using them on Jan 14, 2020

mtrmac commented on Jan 15, 2020

Contributor Author



If I knew this would have been the result I definitely would not have used finalisers. In fact I'm wondering if it would be better to remove them, better safe than sorry?

In general, I think it's nice for the Go bindings to do the work to make the API Go-like, including Go assumptions like no need to worry about memory references/lifetimes.

The code in containers/image that calls these bindings apparently relies on the finalizers, there are no explicit `c1ose` / `Release` calls (I'm actually rather surprised at this... I don't know why I never thought to look for them or call them, my best guess because the `context` / `Data` objects are in-memory-only), and the containers/image library is used in long-term daemons. So, at least for this code, removing finalizers now would lead to undesirable memory leaks unless the callers were adjusted.


That does not necessarily mean that other consumers are the same, but it's a possibility, and nothing in the documentation comments suggests the need to call the `c1ose` / `Release` methods.

  mtrmac deleted the lifetimes branch 3 years ago

  mtrmac mentioned this pull request on Jan 18, 2020



*: Upstream changes from proglottis/gpgme mtrmac/gpgme#2

 Closed

 mtrmac added a commit to mtrmac/image that referenced this pull request on Jan 18, 2020


 Update to github.com/mtrmac/gpgme@v0.1.1 ...

4c7a23f

  mtrmac mentioned this pull request on Jan 18, 2020



Update to github.com/mtrmac/gpgme@v0.1.1 containers/image#794

 Merged

 mtrmac added a commit to mtrmac/docker that referenced this pull request on Feb 20, 2020

 Update to github.com/mtrmac/gpgme v0.1.2 ...

2c7552e

  mtrmac mentioned this pull request on Feb 20, 2020



Update to github.com/mtrmac/gpgme v0.1.2 projectatomic/docker#369

 Open

 mtrmac added a commit to mtrmac/skopeo that referenced this pull request on Feb 20, 2020

 Update to github.com/mtrmac/gpgme v0.1.2 ...

8f7ac72

  mtrmac mentioned this pull request on Feb 20, 2020


Update to github.com/mtrmac/gpgme v0.1.2 containers/skopeo#825

 Merged

 mtrmac added a commit to mtrmac/skopeo that referenced this pull request on Feb 20, 2020


 Update to github.com/mtrmac/gpgme v0.1.2 ...

c48714e

 mtrmac added a commit to mtrmac/skopeo that referenced this pull request on Feb 21, 2020


 Update to github.com/mtrmac/gpgme v0.1.2 ...

29835bb

  mtrmac mentioned this pull request on Feb 21, 2020



Update to github.com/mtrmac/gpgme v0.1.2 containers/skopeo#827

 Merged

 mtrmac added a commit to mtrmac/skopeo that referenced this pull request on Feb 21, 2020


 Update to github.com/mtrmac/gpgme v0.1.2 ...

41dbb0e

  mtrmac mentioned this pull request on Feb 21, 2020

Update to github.com/mtrmac/gpgme v0.1.2 containers/skopeo#828

 Merged

 mtrmac added a commit to mtrmac/skopeo that referenced this pull request on Feb 21, 2020



 Update to github.com/mtrmac/gpgme v0.1.2 ...

0484557

 mtrmac added a commit to mtrmac/machine-config-operator that referenced this pull request on Feb 27, 2020

 Update to github.com/mtrmac/gpgme v0.1.2 ...

ff45426

  mtrmac mentioned this pull request on Feb 27, 2020



Bug 1802871: Update to github.com/mtrmac/gpgme v0.1.2 openshift/machine-config-operator#1519

 Merged

 mtrmac added a commit to mtrmac/machine-config-operator that referenced this pull request on Jun 20, 2020

 Update to github.com/mtrmac/gpgme v0.1.2 ...

59c732d

  mtrmac mentioned this pull request on Jun 20, 2020

Bug 1849298: Update to github.com/mtrmac/gpgme v0.1.2 [4.4] openshift/machine-config-operator#1854

🔗 Merged

🔗 mtrmac added a commit to mtrmac/machine-config-operator that referenced this pull request on Jun 20, 2020

🔗 Update to github.com/mtrmac/gpgme v0.1.2 ...

1bd9475

🔗 mtrmac mentioned this pull request on Jun 20, 2020

Bug 1802880: Update to github.com/mtrmac/gpgme v0.1.2 [4.3] openshift/machine-config-operator#1855

🔗 Closed

🔗 mtrmac added a commit to mtrmac/machine-config-operator that referenced this pull request on Jun 20, 2020

🔗 Update to github.com/mtrmac/gpgme v0.1.2 ...

899f4da

🔗 mtrmac mentioned this pull request on Jun 20, 2020

Bug 1802883: Update to github.com/mtrmac/gpgme v0.1.2 [4.2] openshift/machine-config-operator#1856

🔗 Closed

🔗 mtrmac mentioned this pull request on Jan 19

support passphrase callback mtrmac/gpgme#3

🔗 Closed

Reviewers

👤 proglottis



Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

None yet

2 participants

