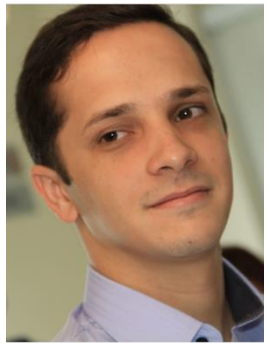
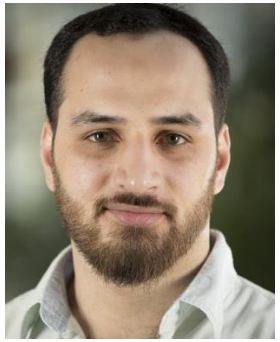
Breaking Type Safety in Go:
An empirical study on the use of the unsafe package

Diego Elias Costa, Suhaib Mujahid, Rabe Abdalkareem, Emad Shihab

Data-driven Analysis of Software

Université Concordia University

The Go programming language

- Major programming language
  - Clean syntax
  - C-like performance
  - Modern language features

kubernetes

docker

- Go has a **strong** and **static** type-system
  - Type-safe by design

Type-safety in GO

```
func main() {
	var i int = 5
	var f float64 = 3.66

	fmt.Println(i + f)

}
```

invalid operation: i + f
(mismatched types int and float64)

Type-safety in GO

```
func main() {
	var i int = 5
	var f float64 = 3.66

	~~fmt.Println(i + f)~~
	fmt.Println(float64(i) + f)
}
```

Go is type-safe…

…unless you use the unsafe package

The unsafe package

- Step around the type-safety of Go programs

```
import "unsafe"

// Pointer arithmetic - (C-style)
p = unsafe.Pointer(uintptr(p) + offset)

// Convert between two types (without compiler checks)
func Float64bits(f float64) uint64 {
        return *(*uint64)(unsafe.Pointer(&f))
}
```

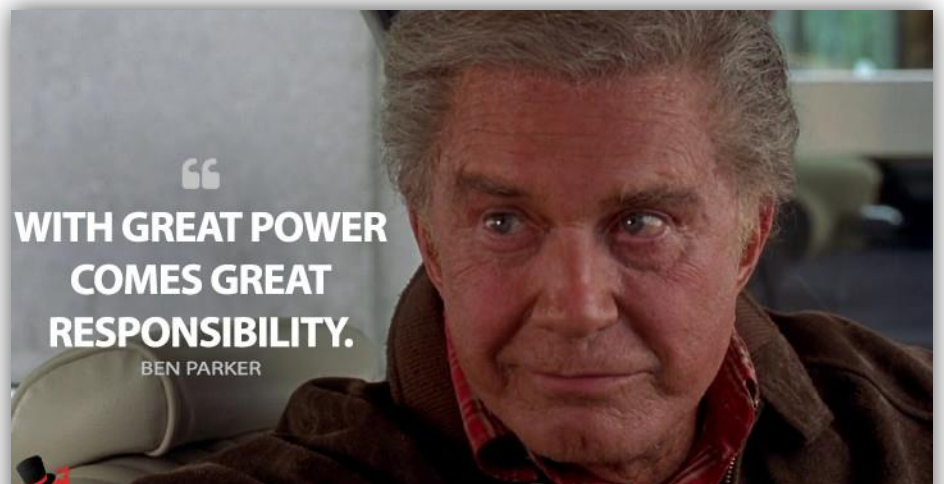
The unsafe package

Pros

- Avoid compiler checks
- Low-level memory manipulation
- Interface with system calls

Cons

- Avoid compiler checks
- Risk of non-portability
- No guarantees of compatibility
- Easy to write bad code



Beware of the unsafe package!

With the unsafe package there are no guarantees.

— "Go Proverbs" by Rob Pike

encoding/gob: can it avoid unsafe? #5159

Closed robpike opened this issue on Mar 29, 2013 · 5 comments

> ***Warning:*** *Avoid* `unsafe` *like the plague; if you can help it.*

[go-nuts] Possible misuse of unsafe.Pointer

Daniel Eloff

Jul 15, 2015 at 2:56 pm

Can someone please verify if my understanding around conversions between uintptr and unsafe.Pointer is correct?

Never use unsafe. Performance is never that critical. – Volker Nov 27 '15 at 10:31

6

First of all, unsafe is usually a bad idea. So is reflection, but unsafe is at least an order of magnitude worse.

Here is your example using pure reflection (http://play.golang.org/p/jTJ6Mhg8q9):

Beware of the unsafe package!

With the unsafe package there are no guarantees.

— "Go Proverbs" by Rob Pike

encoding/gob: can it avoid unsafe? #5159

Closed robpike opened this issue on Mar 29, 2013 · 5 comments

[go-nuts] Possible misuse of unsafe.Pointer

Do Go developers use the unsafe package?

Never use unsafe. Performance is never that critical.

First of all, unsafe is usually a bad idea. So is reflection, but unsafe is at least an order of magnitude worse.

6

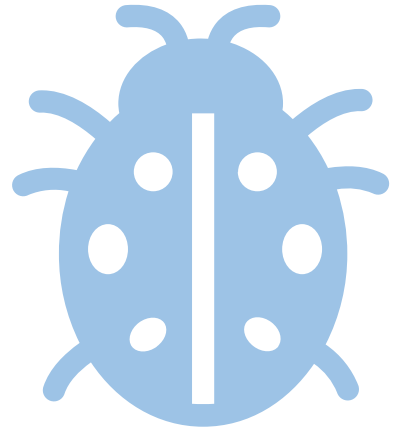
Here is your example using pure reflection (http://play.golang.org/p/jTJ6Mhg8q9):

Studying breaking type-safety in Go

Prevalence?

Why?

Consequences?

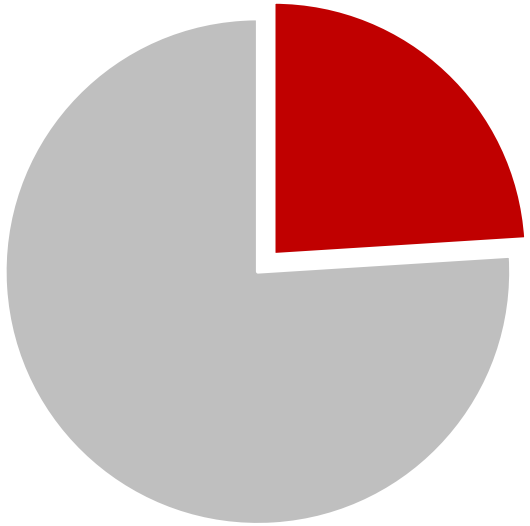


Studied projects

2590 most popular
Go software projects

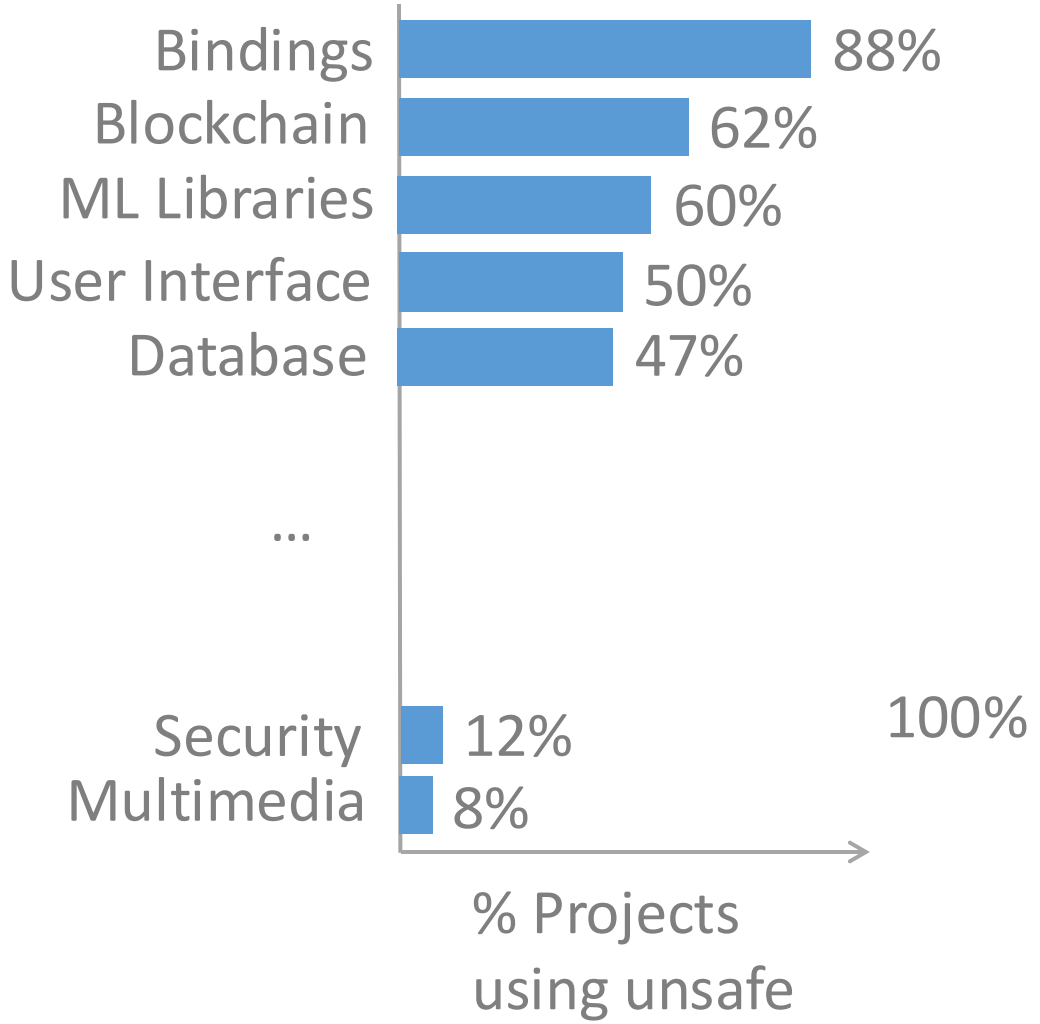
GitHub

Do developers use unsafe?



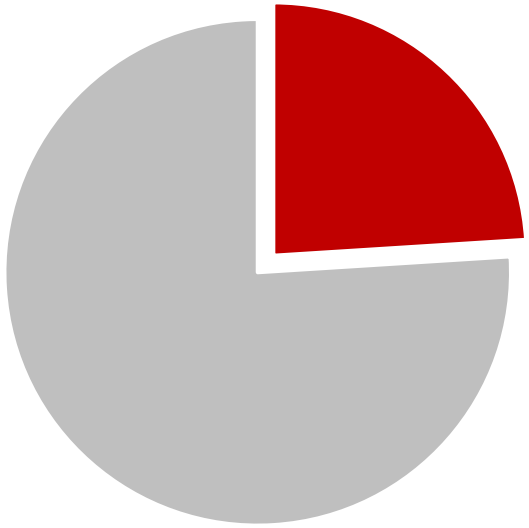
24% of projects use unsafe in their code

What domain of projects rely on unsafe?



All domains

Why developers use unsafe?



We sample **270** projects from the 598 that use unsafe

```
// Pointer arithmetic - (C-style)
p = unsafe.Pointer(uintptr(p) + offset)
```

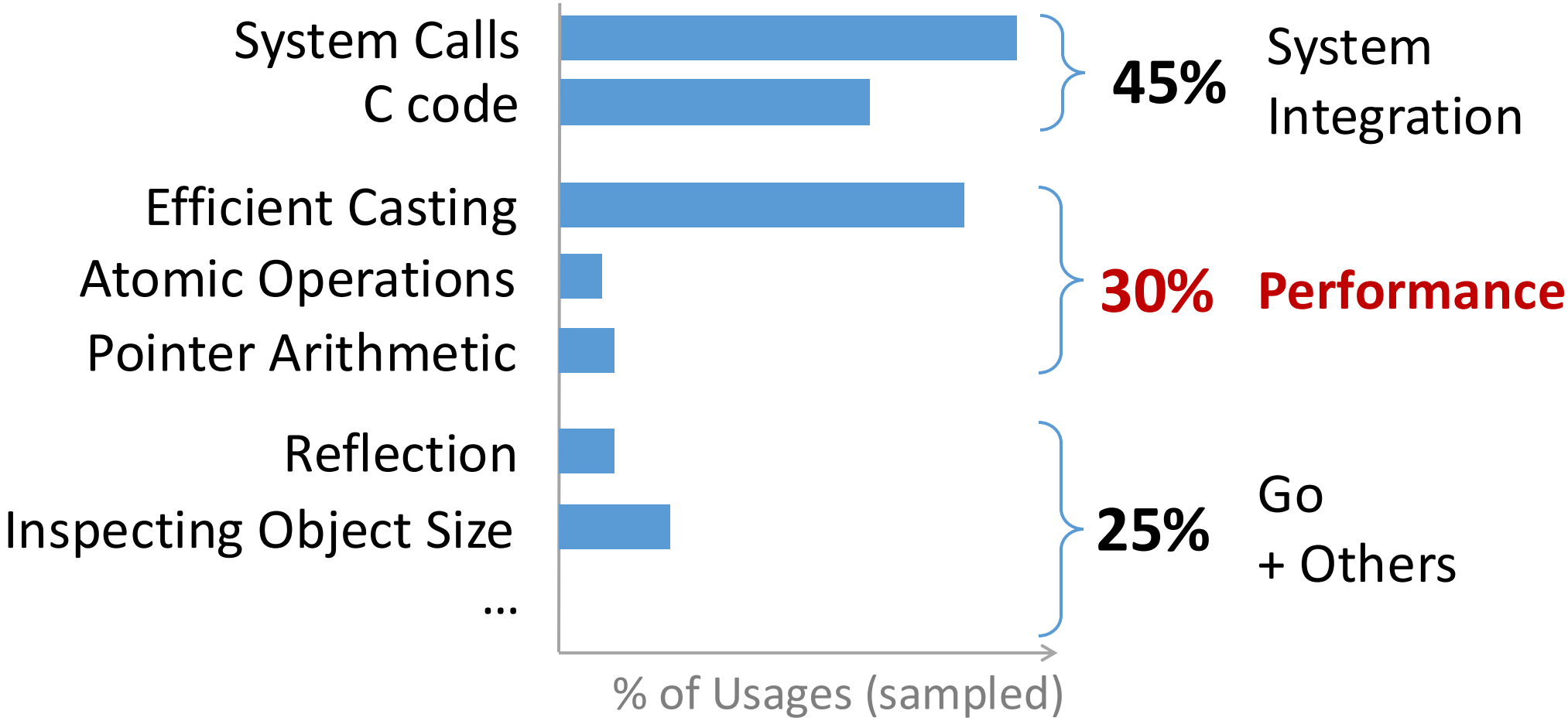
Pointer arithmetic

```
// Convert between two types (without compiler
checks)
func Float64bits(f float64) uint64 {
        return *(*uint64)(unsafe.Pointer(&f))
}
```

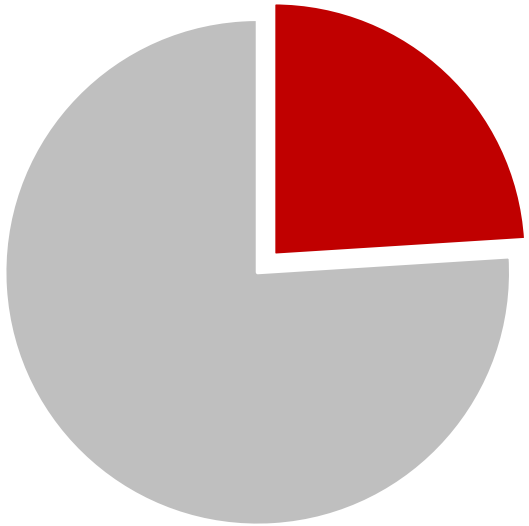
Casting

Manually classify unsafe usages

Why developers use unsafe?



Consequences of unsafe



We look at the pull requests of the 598 projects that use unsafe

Consequences of unsafe

Deployment restriction (20 projects)

> “I wanted to use this package within a Google App Engine project, and due to package ”unsafe” being used, it is **not compatible**”

Consequences of unsafe

Deployment restriction (20 projects)
Runtime errors (16 projects)

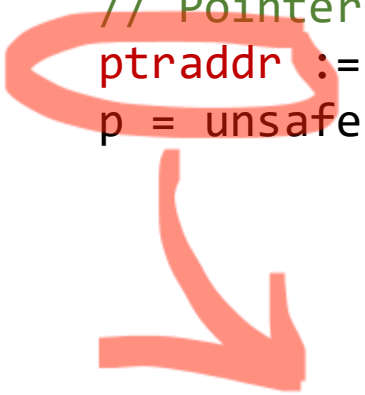
Consequences of unsafe

Deployment restriction (20 projects)
Runtime errors (16 projects)
Wrong API usage (13 projects)

…and the list goes on

Consequences of unsafe

```
// Pointer arithmetic B
ptraddr := uintptr(p)
p = unsafe.Pointer(ptaddr + offset)
```



GC can release `ptraddr` in the middle of the operation!

6.3% of projects that use unsafe have Invalid Pointer conversions

To summarize

Prevalence?

24% of projects use unsafe

Why?

System Integration

Performance Optimization

Consequences?

Higher risk of

- Restrictions
- Runtime errors
- Bugs
- Breakages

Feedback from the Go Team

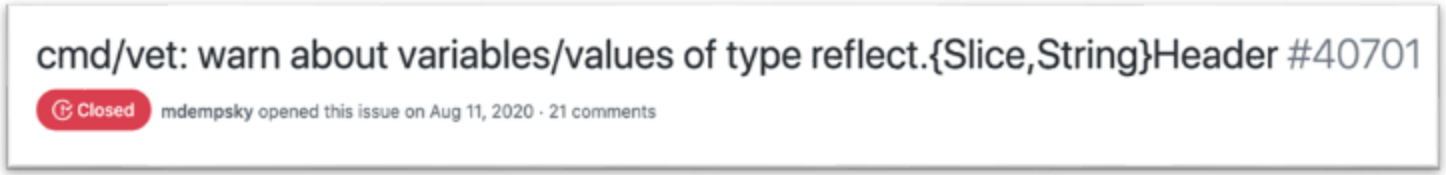
> “Other team members were more optimistic that developers would avoid or could implement their project without using package unsafe.
>
> I think this result will justify **spending more time on making package unsafe easier to use**.”

Matthew Dempsky, maintainer of the GO compiler

Impact on the GO Language

Wrong slice conversion is one of the most common API misuses

New static analysis was released with Go 1.16



Language updates scheduled for Go 1.17

To summarize

Prevalence?

24% of projects use unsafe

Why?

System Integration

Performance Optimization

Consequences?

Higher risk of
- Restrictions
- Runtime errors
- Bugs
- Breakages

Impact on the GO Language

Wrong slice conversion is one of the most common API misuses

New static analysis was released with Go 1.16

cmd/vet: warn about variables/values of type reflect.{Slice,String}Header #40701

Closed mdempsky opened this issue on Aug 11, 2020 · 21 comments

Language updates scheduled for Go 1.17

Go 1.17

unsafe: add Slice(ptr *T, len anyIntegerType) []T #19367

Open mdempsky opened this issue on Mar 2, 2017 · 105 comments

unsafe: add Add function #40481

Open rsc opened this issue on Jul 29, 2020 · 27 comments

@DiegoEliasCosta

diego.costa@concordia.ca

To summarize

Prevalence?

24% of projects use unsafe

Why?

System Integration

Performance Optimization

Consequences?

Higher risk of

- Restrictions
- Runtime errors
- Bugs
- Breakages