Go. Memory-safe language.

Дурнов Алексей Николаевич

Московский физико-технический институт
Физтех-школа радиотехники и компьютерных технологий

Москва, 2025 г.

The development of new product lines for use in service of critical infrastructure or NCFs in a memory-unsafe language (e.g., C or C++) where readily available alternative memory-safe languages could be used is dangerous and significantly elevates risk to national security, national economic security, and national public health and safety.

Автоматическое выделение памяти

```
1  func EscapeAnalysis() *int {
2      // Local var "escape" into heap
3      x := 42
4      return &x
5  }
6
7  func StackAllocation() int {
8      // Stay in stack
9      y := 100
10     return y
11 }
12
```

- Компилятор сам решает где размещать данные
- Нет ручного управления (new/delete)

Проверка границ слайсов

```
1 func CheckBounds() {
2     data := []int{1, 2, 3}
3
4     value := data[5] // panic
5 }
6
```

- Автоматическая проверка границ во время выполнения
- Паника вместо неопределенного поведения
- Возможность восстановления через recover()

Контекст с таймаутом

```
1  func SafeResourceHandling() {
2      ctx, cancel := context.WithTimeout(
3          context.Background(),
4          2*time.Second,
5      )
6      defer cancel()
7
8      req, _ := http.NewRequestWithContext(
9          ctx,
10         "GET",
11         "https://example.com",
12         nil,
13     )
14
15     resp, _ := http.DefaultClient.Do(req)
16     defer resp.Body.Close()
17
18     // Response processing
19 }
20
```

Синхронизация горутин

```
1  func SafeConcurrency() {
2      var wg sync.WaitGroup
3      sharedMap := make(map[int]string)
4      var mu sync.Mutex
5
6      for i := 0; i < 10; i++ {
7          wg.Add(1)
8          go func(idx int) {
9              defer wg.Done()
10             mu.Lock()
11             sharedMap[idx] = fmt.Sprintf("Data %d", idx)
12             mu.Unlock()
13         }(i)
14     }
15     wg.Wait()
16 }
17
```

Прямой доступ к памяти через C

```
1  /*
2  #include <stdlib.h>
3  */
4  import "C"
5  import "unsafe"
6
7  func unsafeAlloc() {
8      ptr := C.malloc(1024)
9      // Memory leak without free
10     // C.free(unsafe.Pointer(ptr))
11
12     // Unsafe conversion
13     data := (*byte)(unsafe.Pointer(ptr))
14     *data = 255
15 }
16
```

- Прямой доступ к сырой памяти через unsafe
- Риск утечек и повреждения памяти
- Потеря всех гарантий безопасности Go

Необработанные задачи

```
1  func WorkerPoolLeak() {
2      jobs := make(chan int, 100)
3
4      // Start workers
5      for i := 0; i < 5; i++ {
6          go func() {
7              for job := range jobs {
8                  process(job)
9              }
10         }()
11     }
12
13     for i := 0; i < 10; i++ {
14         jobs <- i
15     }
16
17     // Forgot to close the jobs channel
18     // The workers continue to wait for the task
19 }
20
```