Safety Not Guaranteed

Using unsafe to syscall Windows APIs without CGO

Walmart Global Tech

Justen Walker;

Principal Software Engineer, Global Tech Platform

Topics

- Short intro to Syscall & unsafe.Pointer
- Loading API Functions
- Calling Windows APIs
  - Mapping C Types to/from Go Types
  - Memory Management (Managed & Unmanaged)
- Examples

WANTED: Sombody to call Win32 API functions from Go with me. This is not a joke. P.O. Box 963, New York, NY 10108. You'll learn how during this talk. Must bring your own Windows. Safety not guaranteed. I have only done this once before.

Early 2016

- K8s
- Mesos/DCOS
- Docker Swarm
- Hashicorp Nomad <- Windows!

Windows Containers 2016

Containers

- In Tech Preview (ie: not production ready)
- Big (in GB)
- Tied to OS version

Alternative: Job Objects

- Apply restrictions to EXEs
- Around since 2003
- Native to all Windows versions

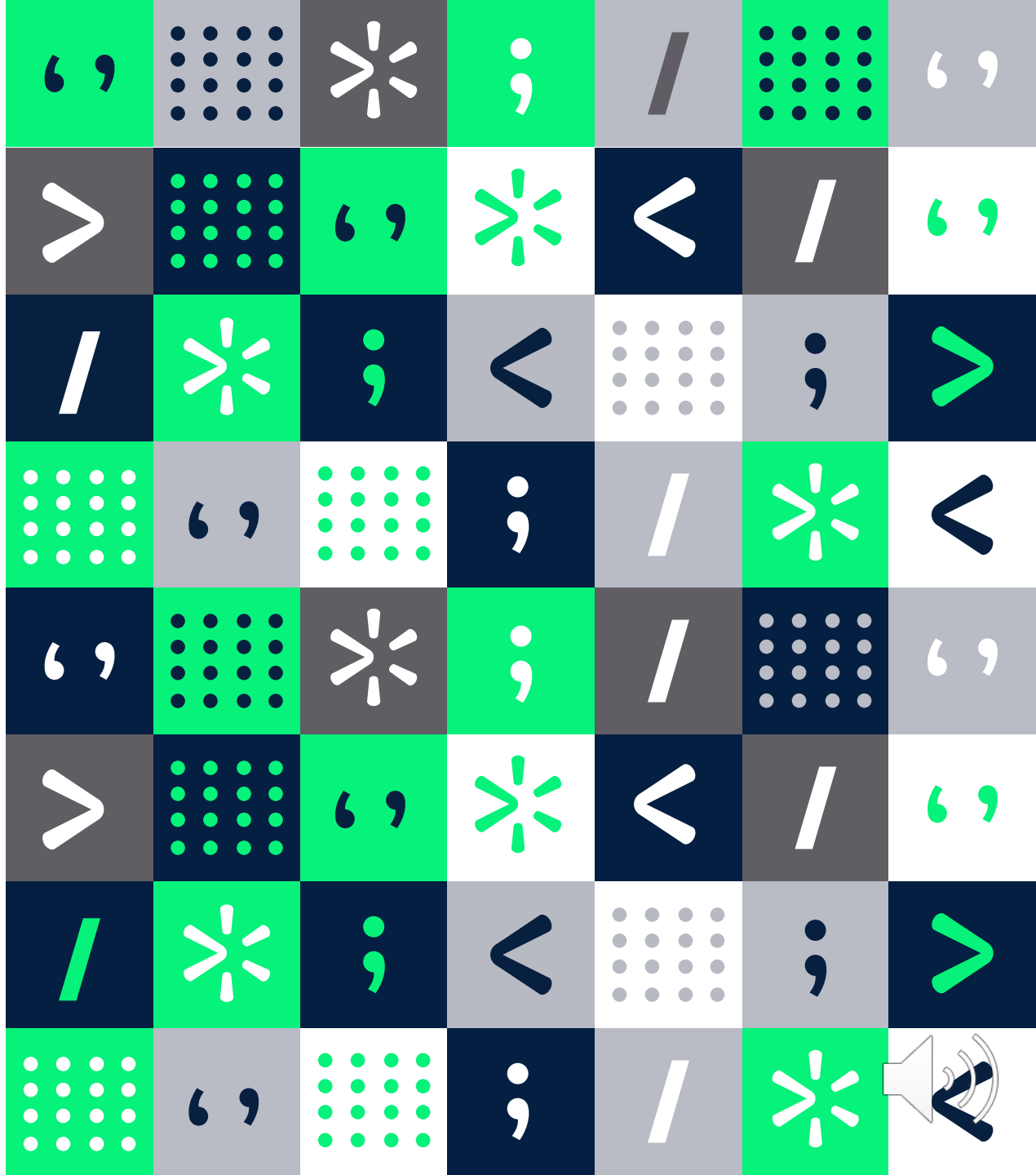
The syscall package

Your gateway to the OS

Global Tech

No “Syscalls” on Windows

- Microsoft does not document Syscalls
- Public API delivered via DLLs
- DLLs require C-like interface



“Syscall must always be guarded with build tags”

- // +build windows
- filename**_windows.go**

The unsafe package

"With the unsafe package, there are no guarantees"

- "Go Proverbs", Rob Pike 2015

unsafe.Pointer

- Type Conversions
  - *T ↔ unsafe.Pointer
  - unsafe.Pointer → uintptr
  - uintptr ⇢ unsafe.Pointer

”reinterpret_cast” of *T1 to *T2

```
type T1 struct {
    a uint32
}
type T2 struct {
    a uint16
    b uint16
}

// sizeof(T1) == sizeof(T2)

var t1 T1

t2 := *(*T2)(unsafe.Pointer(&t1))
```

Pointer Arithmetic

```
data := [...]byte{0x01,0x02,0x03}
p1 := &data[0]
// p1 = 0xc00002c001 => '0x01'

p2 := (*byte)(unsafe.Pointer(uintptr(unsafe.Pointer(&data)) + 2))
// p2 = 0xc00002c003 => '0x03'
```

Pointer Rules

- 6 Rules documented in godoc
- the compiler recognizes these
- Here's the gist
  1. **Never store the result of unsafe.Pointer -> uintptr in a variable**
  2. **Don't point to memory outside allocation boundaries**

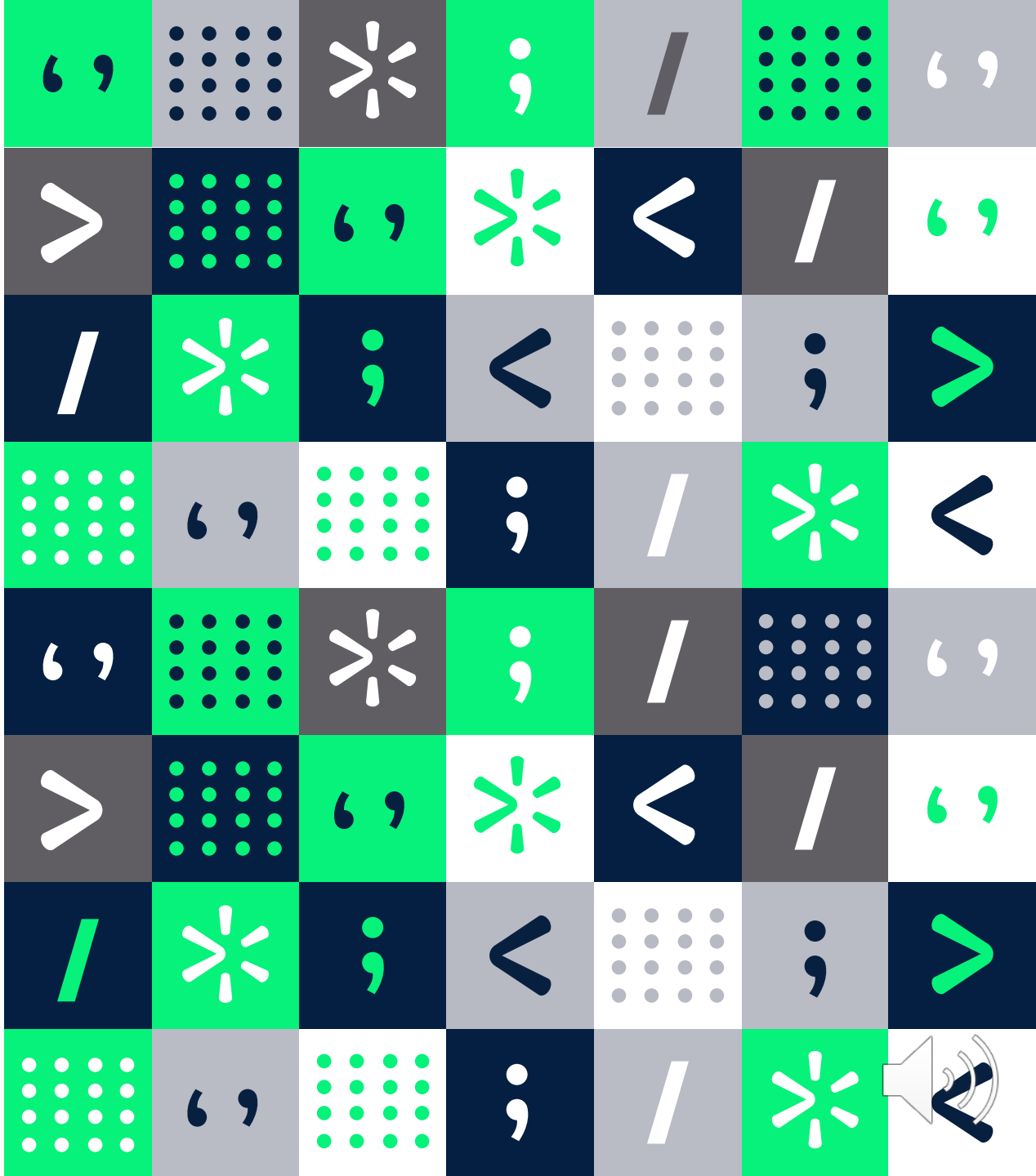
Bad Pointers

```
data := [...]byte{0x01,0x02,0x03}

// INVALID: uintptr stored in var
just_a_number := uintptr(unsafe.Pointer(&data[0]))
p1 := (*byte)(unsafe.Pointer(just_a_number))

// INVALID: pointer past allocated boundary
p2 := (*byte)(unsafe.Pointer(uintptr(unsafe.Pointer(&data[0])) + 3))
```

Loading API Functions

Loading DLLs
Do you need to?

- **golang.org/x/sys/windows**
- Check for the API
- Don’t do work if you don’t have to



Finding DLL and Call Signatures

- docs.microsoft.com
- “Programming reference for the Win32 API”
- Example: CreateJobObject
  - CreateJobObjectA (ANSI)
  - CreateJobObjectW (UTF-16)

CreateJobObjectW function

12/05/2018 • 2 minutes to read

Creates or opens a job object.

Syntax

```
C++

HANDLE CreateJobObjectW(
  LPSECURITY_ATTRIBUTES lpJobAttributes,
  LPCWSTR               lpName
);
```

Library	Kernel32.lib
DLL	Kernel32.dll

Loading a DLL

```
import "syscall"

// Loads kernel32.dll immediately
kernel32DLL = syscall.NewDLL("kernel32.dll")
// Defer loading until a procedure is called
kernel32DLL = syscall.LazyNewDLL("kernel32.dll")

import "golang.org/x/sys/windows"

// (Preferred) same as above, limited to system folders
kernel32DLL = windows.NewSystemDLL("kernel32.dll")
kernel32DLL = windows.NewLazySystemDLL("kernel32.dll")
```

Loading a DLL Procedure

```
import "golang.org/x/sys/windows"
var kernel32DLL = windows.NewLazySystemDLL("kernel32.dll")
var procCreateJobObjectW = kernel32DLL.NewProc("CreateJobObjectW")
```


Calling Windows APIs
Mapping C Types

- **Windows Data Types** reference
- Pointers are uintptr
- float/double use math package.
  (See godoc for **syscall.Proc.Call**)

```
type (
 BOOL          uint32
 BOOLEAN       byte
 BYTE          byte
 DWORD         uint32
 DWORD64       uint64
 HANDLE        uintptr
 HLOCAL        uintptr
 LARGE_INTEGER int64
 LONG          int32
 LPVOID        uintptr
 SIZE_T        uintptr
 UINT          uint32
 ULONG_PTR     uintptr
 ULONGLONG     uint64
 WORD          uint16
)
```

https://docs.microsoft.com/en-us/windows/win32/winprog/windows-data-types

Primitive C-Type mappings

```
type (
    // 'int' is compiler dependent
    // On Windows (32 & 64bit)
    // sizeof(BOOL) = 4
    // typedef int BOOL;
    BOOL   uint32
    // typedef unsigned long DWORD;
    DWORD  uint32
    // typedef PVOID HANDLE;
    HANDLE uintptr
)
```

OpenProcess function

12/05/2018 • 2 minutes to read

Opens an existing local process object.

Syntax

C++

```
HANDLE OpenProcess(
  DWORD dwDesiredAccess,
  BOOL  bInheritHandle,
  DWORD dwProcessId
);
```

Mapping Parameters

- Return HANDLE (uintptr)
- Args
  - *SECURITY_ATTRIBUTES
  - *uint16

CreateJobObjectW function

12/05/2018 • 2 minutes to read

Creates or opens a job object.

Syntax

C++

```
HANDLE CreateJobObjectW(
  LPSECURITY_ATTRIBUTES lpJobAttributes,
  LPCWSTR               lpName
);
```


Go string to C string

```go
// use syscall.BytePtrFromString
func StringToCharPtr(str string) *uint8 {
     // null terminate
    chars := append([]byte(str),0)
    return &chars[0]
}
// use syscall.UTF16PtrFromString
func StringToUTF16Ptr(str string) *uint16 {
    // encode + null terminate
    wchars := utf16.Encode([]rune(str + "\x00"))
    return &wchars[0]
}
```


C String to Go string

```
 1 const wcharsz = uintptr(2) // uint16 = 2 bytes
 2
 3 func UTF16PtrToString(p *uint16) string {
 4     if p == nil { return "" }
 5     end := unsafe.Pointer(p)
 6     var n int
 7     for *(*uint16)(end) != 0 { // at NULL terminator?
 8         end = unsafe.Pointer(uintptr(end) + wcharsz) // advance next char
 9         n++
10     }
11     if n == 0 { return "" }
12     return string(utf16.Decode((*[1 << 30]uint16)(unsafe.Pointer(p))[:n:n]))
13 }
```

Struct Types

- Mapping is pretty much 1-to-1, Repeat for composed types

```
// typedef struct _SECURITY_ATTRIBUTES {
type SecurityAttributes struct {
//   DWORD  nLength;
    Length             uint32
//   LPVOID lpSecurityDescriptor;
    SecurityDescriptor *SECURITY_DESCRIPTOR
//   BOOL   bInheritHandle;
    InheritHandle      uint32
}
// } SECURITY_ATTRIBUTES, *PSECURITY_ATTRIUBTES
//   *LPSECURITY_ATTRIBUTES;
```


Calling Windows APIs – Go func to DLL call

```
func OpenProcess(desiredAccess uint32, inheritHandle bool, processId uint32) (syscall.Handle, error) {
    var bInheritHandle uint32
    if inheritHandle {
        bInheritHandle = 1
    }
    handle, _, errno := procOpenProcess.Call(
        uintptr(desiredAccess),
        uintptr(bInheritHandle),
        uintptr(processId))
    if handle == 0 {
        if errno != 0 {
            return 0, errno
        }
    }
    return syscall.Handle(handle), nil
}
```

Inspecting Return Value

Return value

If the function succeeds, the return value is an open handle to the specified process.

If the function fails, the return value is NULL. To get extended error information, call GetLastError.

Remarks

...

When you are finished with the handle, be sure to close it using the CloseHandle function.

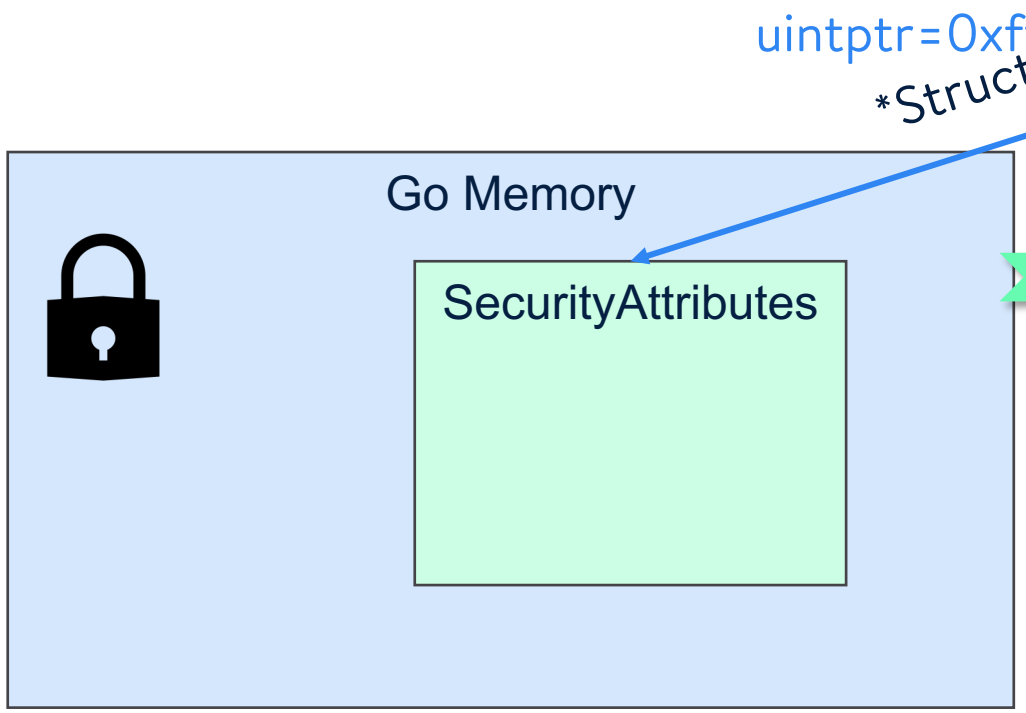
CreateJobObject

- Pass Go Pointers to C with care
- Go compiler helps here

```
func CreateJobObject(attr *SecurityAttributes, name string) (syscall.Handle, error) {
    ret, _, err := procCreateJobObjectW.Call(
        uintptr(unsafe.Pointer(attr)),
        uintptr(unsafe.Pointer(syscall.UTF16PtrFromString(name))),
    )
    if ret == 0 { // ret == NULL
        return 0, err
    }
    return syscall.Handle(ret), nil
}
```

Locking memory down during syscall

- (4) Conversion of a Pointer to a uintptr when calling syscall.Syscall




```
func CreateJobObject(attr *SecurityAttributes,
    ret, _, err := procCreateJobObjectW.Call(
        uintptr(unsafe.Pointer(attr)),
        uintptr(unsafe.Pointer(syscall.UTF16Ptr
    )
    if ret == 0 { // ret == NULL
        return 0, err
    }
    return syscall.Handle(ret), nil
}
```

golang.org/x/sys/windows/mkwinsyscall

- Parses files for lines with //sys
- Interprets parameter lists
- Generates function wrappers
- Knows some simple conversions:
  - bool to BOOL
  - string to C-String/UTF-16 String
  - Slice to Array

mkwinsyscall - source

main.go

```
package mysyscalls

//go:generate go run golang.org/x/sys/windows/mkwinsyscall -output zsyscalls_windows.go
syscalls_windows.go
```

syscalls_windows.go

```
package mysyscalls

//sys   CreateJobObject(jobAttr *SecurityAttributes, name *uint16) (handle Handle, err error) =
kernel32.CreateJobObjectW
//sys   GetStdHandle(stdhandle uint32) (handle Handle, err error) [failretval==InvalidHandle]
```

zsyscalls_windows.go (generated)

```go
package mysyscalls

//...snip

var (
  modkernel32 = windows.NewLazySystemDLL("kernel32.dll")

  procCreateJobObjectW = modkernel32.NewProc("CreateJobObjectW")
)

func CreateJobObject(jobAttr *SecurityAttributes, name *uint16) (handle Handle, err error) {
  r0, _, e1 := syscall.Syscall(procCreateJobObjectW.Addr(), 2, uintptr(unsafe.Pointer(jobAttr)),
uintptr(unsafe.Pointer(name)), 0)
  handle = Handle(r0)
  if handle == 0 {
    if e1 != 0 {
      err = errnoErr(e1)
    } else {
      err = syscall.EINVAL
    }
  }
  return
}
```

Memory Management

Unmanaged

Syscall allocates memory and transfers ownership to caller

```
BOOL CredEnumerateW(
  LPCWSTR      Filter,
  DWORD        Flags,
  DWORD        *Count,
  PCREDENTIALW **Credential
);
```

Caller Managed

Caller allocates memory and Syscall writes to it

```
DWORD GetExtendedTcpTable(
  PVOID            pTcpTable,
  PDWORD           pdwSize,
  BOOL             bOrder,
  ULONG            ulAf,
  TCP_TABLE_CLASS  TableClass,
  ULONG            Reserved
);
```

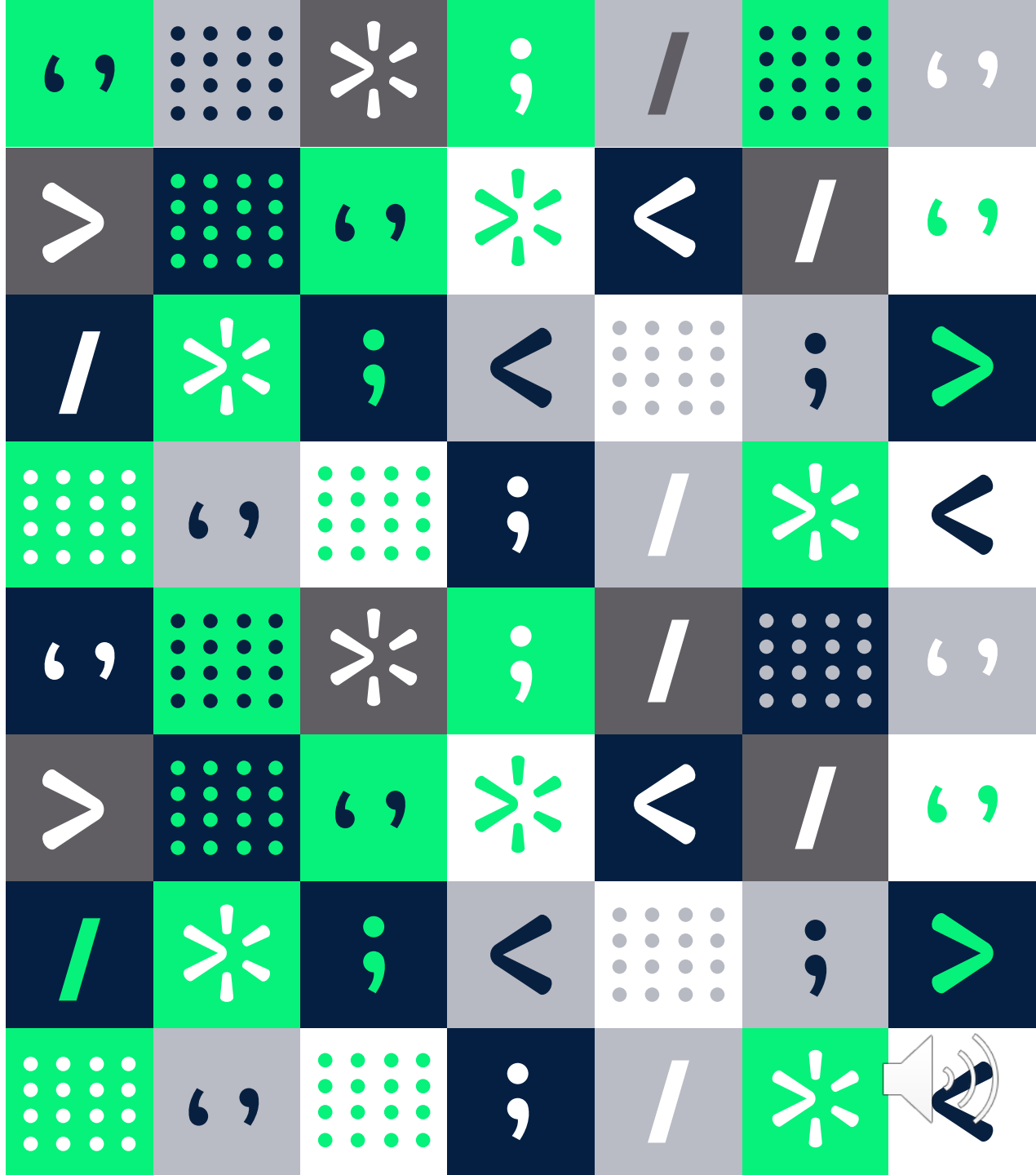
Unmanaged Memory: CredEnumerateW

```
BOOL CredEnumerateW(
  LPCWSTR      Filter,
  DWORD        Flags,
  DWORD        *Count,
  PCREDENTIALW **Credential
);
```

Pointer to an array of pointers to credentials. The returned credential is a single allocated block. Any pointers contained within the buffer are pointers to locations within this single allocated block. The single returned buffer must be freed by calling CredFree.

When to Free?

- Use directly, Free when finished
- Copy and Free Immediately



Example: Designing for Unmanged Memory

```
1 //sys CredEnumerateW(filter *uint16, flags uint32, count *uint32, credentials ***_CREDENTIALW) (err
  error) [failretval==0] = advapi32.CredEnumerateW
2 //sys CredFree(buffer unsafe.Pointer)  = advapi32.CredFree
3
4 func CredEnumerate() (*Credentials, error) {
5     var cred Credentials
6     var err := CredEnumerateW(nil, CRED_ENUMERATE_ALL_CREDENTIALS, &cred.count, &cred.items)
7     if err != nil {
8         return nil, err
9     }
```

Pointer to an array of pointers to credentials. The returned credential is a single allocated block. Any pointers contained within the buffer are

```
15     count uint32
16     items **_CREDENTIALW
17 }
```

Example: Designing For Unmanged Memory - Free

```
type Credentials struct {
    free  bool
    count uint32
    items **_CREDENTIALW
}

func (c *Credentials) Free() {
    if c.free { return }
    CredFree(unsafe.Pointer(c.items))
    c.free = true
}
```


Example: Designing For Unmanged Memory - Iterating

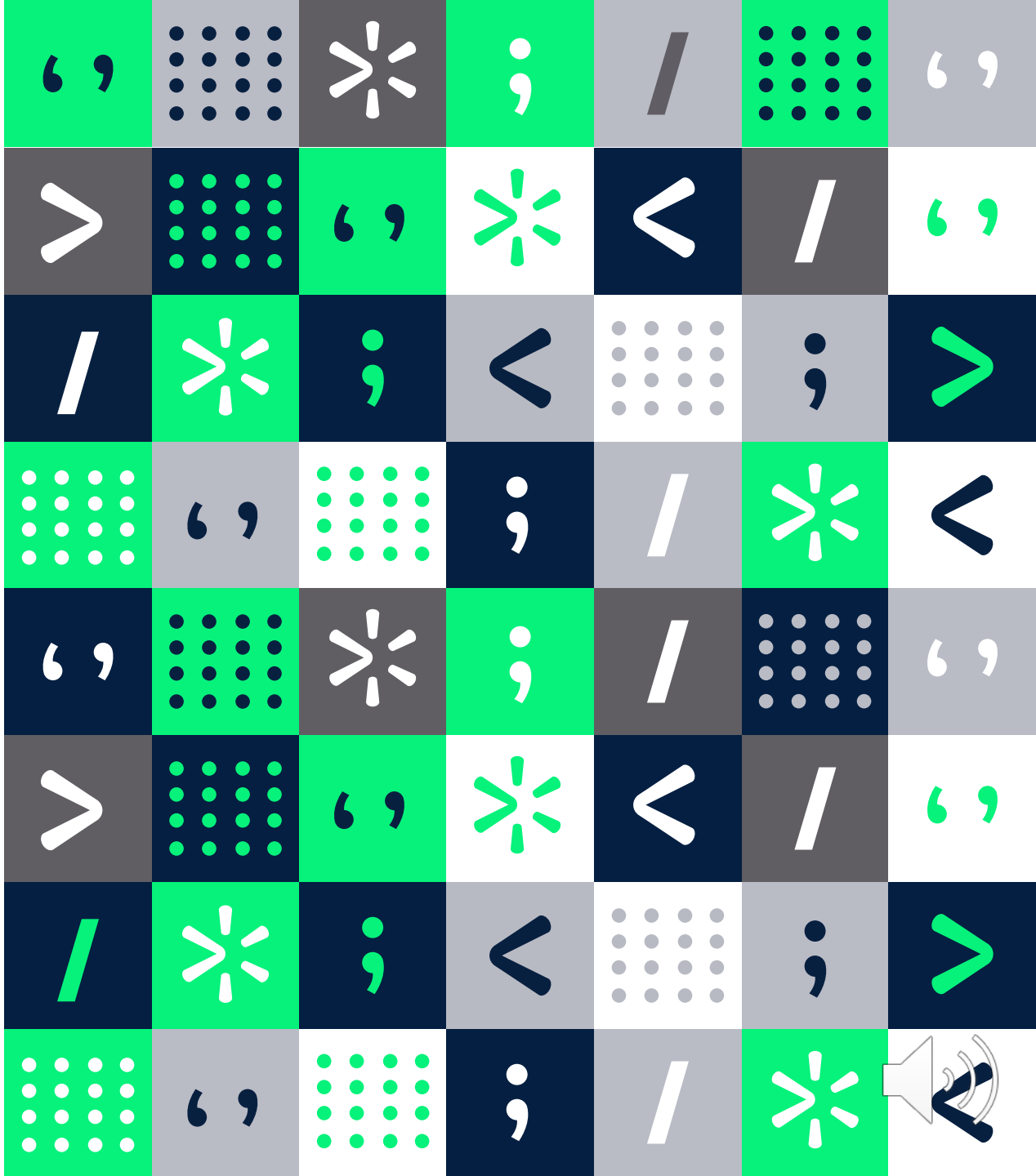
```go
func (c *Credentials) ForEach(fn func(cred Credential) error) error {
    if c.free { return errors.New("memory freed") }
    var sz = unsafe.Sizeof(&_CREDENTIALW{})
    for i := uint32(0); i < c.count; i++ {
        pcred := *(**_CREDENTIALW)(unsafe.Pointer(uintptr(unsafe.Pointer(c.items)) + uintptr(i)*sz))
        if err := fn(Credential{
            free: &c.free,
            pcred: pcred,
        }); err != nil {
            return err
        }
    }
    return nil
}

type Credential struct {
    free  *bool
    pcred *_CREDENTIALW
}

func (c *Credential) Name() string {
    if *c.free { return "" }
    return utf16PtrToString(c.pcred.TargetName)
}

// ... and other "Getters"
```


Improvement – Copy & Free

More difficult to misuse

Safer

Easier to code against



Example: Copy & Free Design - CredEnumerate

```go
func CredEnumerate() ([]Credential, error) {
    var count uint32
    var items **_CREDENTIALW
    err := CredEnumerateW(nil, CRED_ENUMERATE_ALL_CREDENTIALS, &count, &items)
    if err != nil {
        return nil, err
    }
    defer CredFree(unsafe.Pointer(items)) // Free Immediately before returning
    var sz = unsafe.Sizeof(&_CREDENTIALW{})
    creds := make([]Credential, 0, int(count))
    for i := uint32(0); i < count; i++ {
        pcred := *(**_CREDENTIALW)(unsafe.Pointer(uintptr(unsafe.Pointer(items)) + uintptr(i)*sz))
        // toCredential: converts *_CREDENTIALW to Credential by copying
        creds = append(creds, toCredential(pcred))
    }
    return creds, nil
}
```

Example: Copy & Free Design – Credential/Attribute

```
 1 type Credential struct {
 2     Name         string
 3     Alias        string
 4     Type         string
 5     Comment      string
 6     UserName     string
 7     Credential   []byte
 8     LastWritten time.Time
 9     Attributes   []CredentialAttribute
10 }
11
12 type CredentialAttribute struct {
13     Keyword string
14     Flags   uint32
15     Value   []byte
16 }
```

Managed Memory: Generic Example

Generic Call Pattern

```
var (buflen int;buffer []byte;err error;ret uintptr)
for {
    ret, _, err = procSomeWindowsAPI.Call(
        uintptr(unsafe.Pointer(&buffer[0])),
        uintptr(unsafe.Pointer(&buflen)),
    )
    if err == syscall.ERROR_INSUFFICIENT_BUFFER {
        buffer = make([]byte, buflen)
        continue
    }
    break
}
result := (*WinAPIGoStruct)(unsafe.Pointer(&buffer[0])) // check ret before this
```

Managed Memory: GetExtendedTcpTable

```
DWORD GetExtendedTcpTable(
  PVOID           pTcpTable,
  PDWORD          pdwSize,
  BOOL            bOrder,
  ULONG           ulAf,
  TCP_TABLE_CLASS TableClass,
  ULONG           Reserved
);
```

`ulAf`

The version of IP used by the TCP endpoints.

Value	Meaning
AF_INET	IPv4 is used.
AF_INET6	IPv6 is used.

`TableClass`

The type of the TCP table structure to retrieve. This parameter can be one of the values from the TCP_TABLE_CLASS enumeration.

The TCP_TABLE_CLASS enumeration value is combined with the value of the *ulAf* parameter to determine the extended TCP information to retrieve.

Example: GetExtendedTcpTable – Syscall Loop

```
1 //sys _GetExtendedTcpTable(tcpTable *byte, tableSize *uint32, order bool, ulAf uint32, tableClass
  _TCP_TABLE_CLASS, reserved uint32) (ret syscall.Errno)  = iphlpapi.GetExtendedTcpTable
2
3 func GetTCPTable() ([]TableRow, error) {
4     var tableSize uint32
5     var tcpTable []byte
6     // query for the buffer size by sending null/0
7     ret := _GetExtendedTcpTable(nil, &tableSize, true, _AF_INET4, _TCP_TABLE_OWNER_PID_ALL, 0)
8     for {
9         if ret == 0 { break }
10        if ret == _ERROR_INSUFFICIENT_BUFFER {
11            tcpTable = make([]byte, int(tableSize))
12            ret = _GetExtendedTcpTable(&tcpTable[0], &tableSize, true, _AF_INET4,
  _TCP_TABLE_OWNER_PID_ALL, 0)
13            continue
14        }
15        return nil, ret
16    }
17    return convertToTableRows(tcpTable), nil
18 }
```

Example: GetExtendedTcpTable - []byte to Table

```
type TableRow struct {
    Local  *net.TCPAddr
    Remote *net.TCPAddr
    State  TcpState
    PID    int
}

func convertToTableRows(raw []byte) (result []TableRow) {
    type _MIB_TCPTABLE_OWNER_PID struct {
        NumEntries uint32
        Table      [1]_MIB_TCPROW_OWNER_PID
    }

    table := (*_MIB_TCPTABLE_OWNER_PID)(unsafe.Pointer(&raw[0]))
    n := int(table.NumEntries)
    if n == 0 { return }
    // ...
```

C++

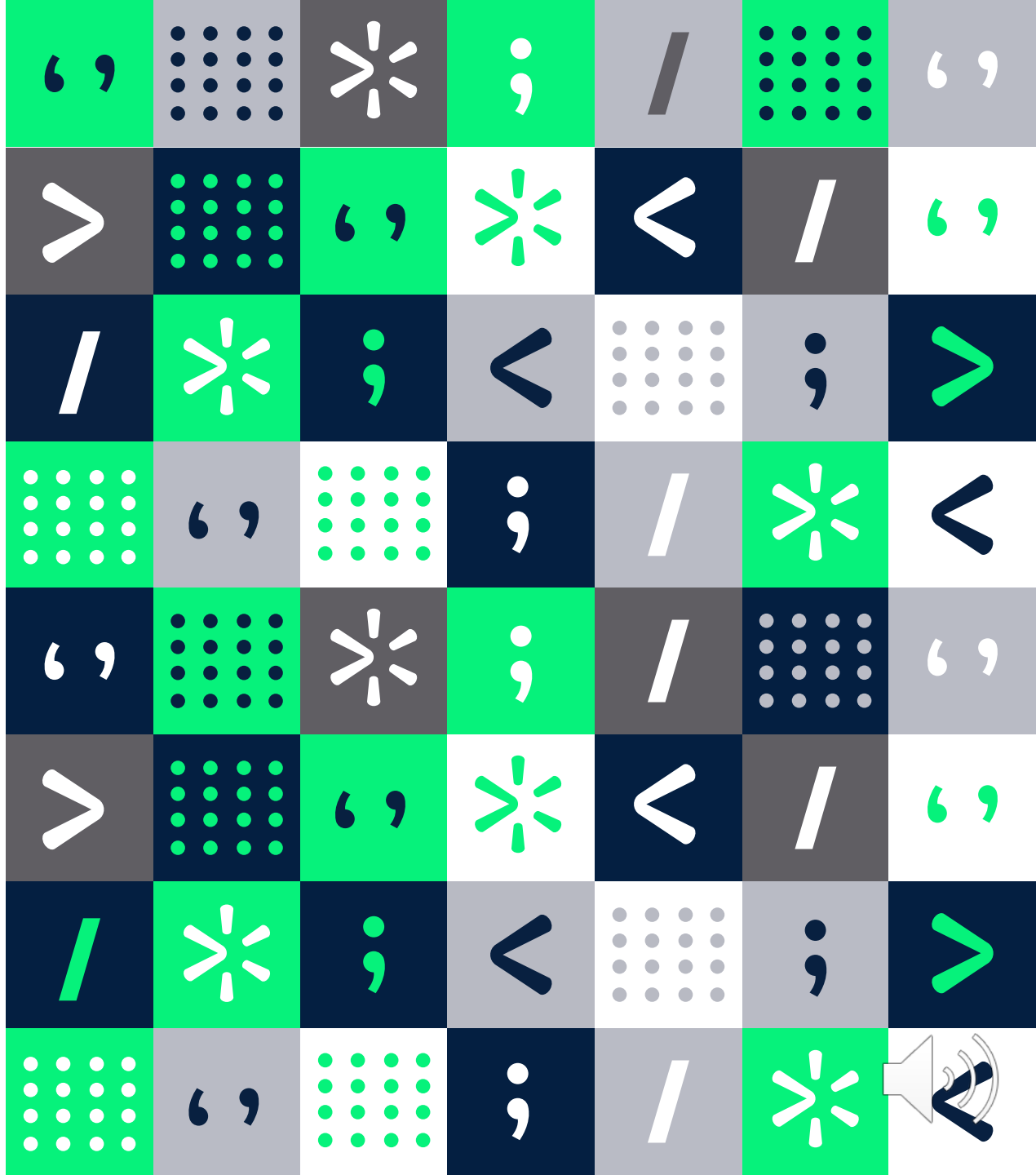
```
typedef struct _MIB_TCPTABLE_OWNER_PID {
  DWORD                dwNumEntries;
  MIB_TCPROW_OWNER_PID table[ANY_SIZE];
} MIB_TCPTABLE_OWNER_PID, *PMIB_TCPTABLE_OWNER_PID;
```

Example: GetExtendedTcpTable - Table to Rows

```
// func convertToTableRows(raw []byte) (result []TableRow) { ...
    result = make([]TableRow, n)
    rows := (*[1 << 30]_MIB_TCPROW_OWNER_PID)(unsafe.Pointer(&table.Table))[0:n:n]
    for i, row := range rows {
        result[i] = TableRow{
            Local:  convertToTCPv4Addr(row.LocalAddr, row.LocalPort),
            Remote: convertToTCPv4Addr(row.RemoteAddr, row.RemotePort),
            PID:    int(row.OwningPID),
            State:  TcpState(row.State),
        }
    }
    return
}
```

Full Code Available

github.com/justenwalker/
gophercon-2020-winapi



Thank you!();