

Golang and C++ interoperability

25th January 2019

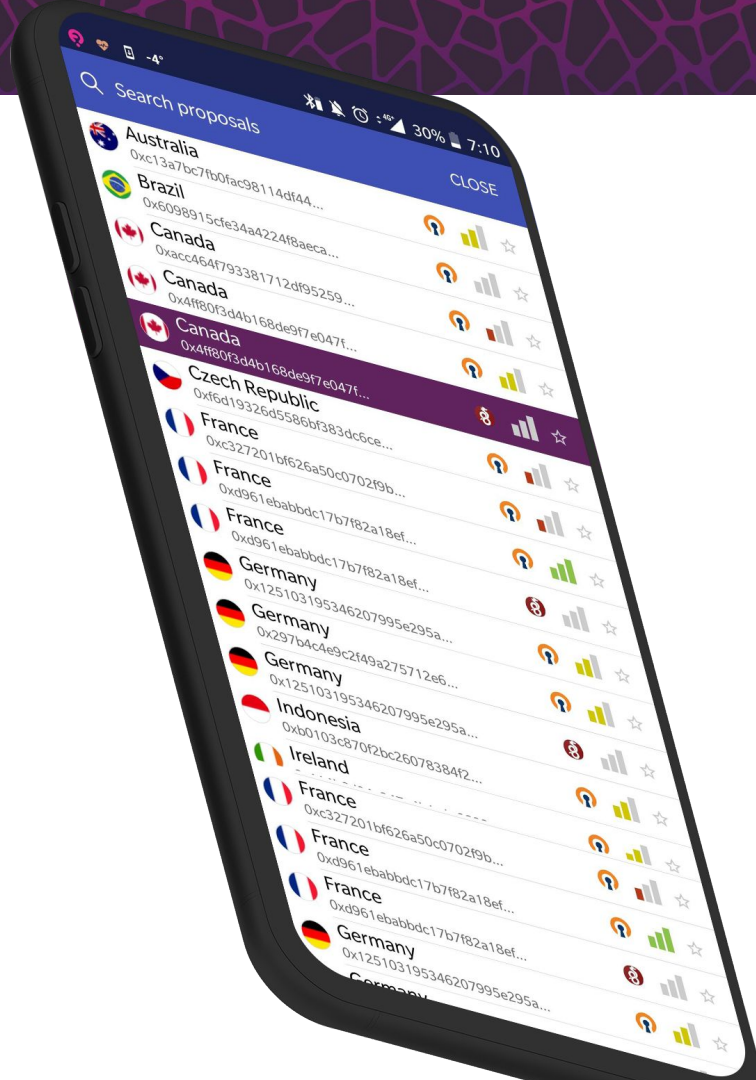


Tadas Valiukas

Tadas Valiukas - one of the main contributors @ Mysterium Network - comes with 14 years of bug making experience.

He began his journey as a C/C++ developer before discovering the “amazing” world of enterprise java.

A year ago he switched primarily to golang and hasn't looked back since.



WHAT IS MYSTERIUM NETWORK?

World's first decentralized VPN.

MysteriumVPN Alpha

Available on Mac, Windows and Android:

<https://mysterium.network/vpn/download-alpha/>

Node Runner Alpha

Hear first about node incentives and our node runner alpha program:

<https://mysterium.network/node-runners/run-test-node/>

OUR TECHNOLOGY STACK:

- Golang ❤️❤️
- JavaScript (because #modern)
- Python (because #startup)
- Bash (because #travis)
- Solidity for smart contracts (Ethereum to 🐦 ○)
- C/C++ (because #time_tested_classics)

TODAY WE WILL COVER:

PROBLEM

Historically, nodes launch external preconfigured Openvpn process for each connection.

This is:

- Slow
- Hard to provision
- Hard to distribute (single exe anyone?)
- External dependency
- Super complicated on mobile devices

IDEA

Integrate openvpn functionality into Mysterium Network node.

SOLUTION

Openvpn + go =
openvpn-go package
Available at:
<https://github.com/mysteriumnetwork/go-openvpn>

FIRST STEP IN A LONG JOURNEY

- Openvpn3 - official OpenVpn C++ library:
 - Client (or connector) only
 - Written in C++
 - Runs (or compiles) in all major platforms (or at least claimed)
- Problems:
 - Golang doesn't talk to C++ directly
 - Super complicated build process on each platform
 - Depends on bunch of other C/C++ projects (openssl, asio, lzo)
- Solution - build a static C compatible library and “go gettable” package:
 - tiny C layer to export C style functions around C++ code
 - Static precompiled openvpn3 library included
 - Go package with pure go interaction for caller

Crash course of C program build process

- **Compilation phase:**
 - Compiles each source (.c / .c++) file into object file (.o) independently
- **Linking phase:**
 - Links or copies all object files into final executable or shared library
 - Resolves all external dependencies (copies static library code, generates import table for runtime libs)
- **Definitions:**
 - Static library - a bunch of object files collected into archive (.a) ([no link phase dependency resolution!](#))
 - Shared library - distributable binary file with exported functionality, loadable at runtime. Without entrypoint
 - Executable - executable binary which depends only on shared libraries with entry point.

C + go → Cgo

```
1  package main
2  /*
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  static void printme(char* s) {
7      printf("%s\n",s);
8  }
9  */
10 import "C"
11
12 ► func main() {
13     cStr:=C.CString("hello world")
14     C.printme(cStr)
15 }
16
```


LONG FORGOTTEN FOREST OF CALLBACKS

- Openvpn3 library loves callbacks:
 - Logging callback func
 - State events callback func
 - Statistics callback func (bytes in, bytes out)
 - External tunnel setup factory (24 functions!)
- Our goal - pure go API for openvpn3 library
- Golang hates idea of passing go function refs to C code
- Possible ways:
 - User defines C functions of it's own - yuck. X
 - User defines special static go functions (`//export ...`) accessible from C and passes them. X
 - Callback registry pattern (<https://github.com/golang/go/wiki/cgo>). V

A SWAMP OF STATIC LIBRARIES

- From source to final binary:
 - Compilation phase - any source (.c .cpp) compiled to object file (*.o)
 - Linking phase - bunch of object files linked into final binary (.exe or .dll)
- Static library (*.a) - a bunch of compiled object files (*.o) thrown together
- First surprise - openvpn3 static library ended up without any of its dependencies
- Solution - take openvpn3 and its deps (static libs), extract them and concat to a single static lib

FORGOTTEN C++ RUNTIME

- Cgo uses c compiler (and linker) by default
- Openvpn3 is written in C++ with refs to C++ runtime
- A lot of funny errors on “go build” execution
- Problem - how to ask cgo to add c++ runtime on link phase
 - Gcc -lstdc++ doesn't always help
- Dummies to the rescue!:
 - Place an empty (dummy) .cpp file in your go package
 - Cgo sees cpp file and calls c++ linker at the link phase!

```
package main

import (
    "fmt"
    "strings"

    "github.com/mysteriumnetwork/go-openvpn/openvpn3"
)

type openvpnCallbacks struct {
}

func (lc *openvpnCallbacks) Log(text string) {
    lines := strings.Split(text, sep: "\n")
    for _, line := range lines {
        fmt.Println( a...: "Openvpn log >>", line)
    }
}

func (lc *openvpnCallbacks) OnEvent(event openvpn3.Event) {
    fmt.Printf( format: "Openvpn event >> %v\n", event)
}

func (lc *openvpnCallbacks) OnStats(stats openvpn3.Statistics) {
    fmt.Printf( format: "Openvpn stats >> %v\n", stats)
}

func main() {
    config := openvpn3.NewConfig( profile: "Contents of standard openvpn profile")

    session := openvpn3.NewSession(
        config,
        openvpn3.UserCredentials{
            Username: "openvpn username",
            Password: "openvpn password",
        },
        &openvpnCallbacks{},
    )

    session.Start()

    //session.Stop() - if you need to end it

    err := session.Wait()
    if err != nil {
        fmt.Println( a...: "Openvpn3 error: ", err)
    } else {
        fmt.Println( a...: "Graceful exit")
    }
}
```












FULLY FUNCTIONAL OPENVPN CLIENT IN GO



ACHIEVEMENT UNLOCKED

Fully functional Openvpn package....

...ON SINGLE MACOS (x64) ARCHITECTURE

Branch: master ▾ go-openvpn / openvpn3 / bridge /		
 zolia use openvpn3 reconnect		
..		
 libopenvpn3_android_amd64.a	use openvpn3 reconnect	
 libopenvpn3_android_arm64.a	use openvpn3 reconnect	
 libopenvpn3_android_armeabi-v7a.a	use openvpn3 reconnect	
 libopenvpn3_android_x86.a	use openvpn3 reconnect	
 libopenvpn3_darwin_amd64.a	use openvpn3 reconnect	
 libopenvpn3_ios_arm64.a	use openvpn3 reconnect	
 libopenvpn3_linux_amd64.a	use openvpn3 reconnect	
 libopenvpn3_windows_amd64.a	use openvpn3 reconnect	

It's always easier to follow a known path - let's cross compile everything!

Result:

- 3 major oses (win, osx, linux), amd64 architecture
- Mobile platforms (ios, android), both arm7, arm64 and simulator support
- ... and separate lib for each architecture and OS combination 📁

LOOKS COOL...BUT IS IT USEFUL?



How it actually ended (on Android):

C++ library...

... wrapped in C...

... imported as Go package inside our node...

... which itself exported as native java shared library...

... with additional functionality with Kotlin ...

... loaded in JVM at runtime...

... and it works!

QUICK RECAP:

- Distribution of static libs inside go package
- Obfuscated code problem (some users simply reject not fully transparent packages)
- Maintenance - tracking of upstream patches
- Works on my machine syndrome (although solvable)
- iOS framework problem
- Golang philosophy - dist code not bin

THANK YOU. QUESTIONS?