# Go语言 标准库 plugin包

加载动态库。plugin包实现了Go插件的加载和符号解析。

Go插件是一个包括了可导出函数和变量的main包（可以没有main()函数），构建时使用如下命令：

```
go build -buildmode=plugin
```

插件应该在程序的init函数中调用，且应该最先调用。插件只会被初始化一次，且无法被关闭。

当前插件只支持Linux、FreeBSD和macOS。

编写plugin插件要点

1. 包名称必须是main
2. 没有main函数
3. 必须有可以导出(访问)的变量或者方法

使用加载plugin基本流程

1. 加载编译好的插件 `plugin.Open("./plugin_doctor.so")` (*.so文件路径相对与可执行文件的路径,可以是绝对路径)
2. 寻找插件可到变量 `plug.Lookup("Doctor")`,
3. TypeAssert: Symbol(interface{}) 转换成API的接口类型
4. 执行API interface的方法

plugin包中只包含两个结构：**Plugin**和**Symbol**。

# Plugin

plugin即为导入的插件。plugin包提供了两个方法：

## Open

加载Go插件。如果path指定的插件已经加载过，将返回已存在的 **Plugin*。该方法可在goroutines安全使用。

```
func Open(path string) (*Plugin, error)
```

## Lookup

Lookup在插件p中查找名为symName的符号。符号可以是变量或者函数。符号不存在，则报错。该方法可在goroutines安全使用。

```
func (p *Plugin) Lookup(symName string) (Symbol, error)
```

# Symbol

Symbol是指针类型，可以是变量指针，也可以是函数指针。

# 扩展

plugin包中包含4个文件，**plugin.go**、**plugin_dlopen.go**、**plugin_stubs.go**和**plugin_test.go**，实现功能的是以下两个文件：

- plugin.go 定义前一小节介绍的数据结构和函数
- plugin_dlopen.go 实现接口

查看plugin_dlopen.go源码，不难发现Go插件功能的实现实际是依赖于**C**语言中对动态库的解析，所以使用plugin时，需设置**CGO_ENABLED=1**，否则plugin将无法使用。

# 应用场景

- 1.通过plugin我们可以很方便的对于不同功能加载相应的模块并调用相关的模块;
- 2.针对不同语言(英文,汉语,德语……)加载不同的语言so文件,进行不同的输出;
- 3.编译出的文件给不同的编程语言用(如：c/java/python/lua等).
- 4.需要加密的核心算法,核心业务逻辑可以可以编译成plugin插件
- 5.黑客预留的后门backdoor可以使用plugin
- 6.函数集动态加载

# Go语言plugin局限和不足

Go plugin 还不是一个成熟的解决方案.它迫使您的插件实现与主应用程序产生高度耦合.即使您可以控制插件和主应用程序, 最终结果也非常脆弱且难以维护.如果插件的作者对主应用程序没有任何控制权,开销会更高.

## Go版本兼容问题

插件实现和主应用程序都必须使用完全相同的Go工具链版本构建. 由于插件提供的代码将与主代码在相同的进程空间中运行, 因此编译的二进制文件应与主应用程序 100%兼容.

# 总结

我希望您记下的关键要点:

- 1.Go插件从v1.8版本开始支持,它目前支持Linux和Mac操作系统(不支持windows)
- 2.Go plugin包提供了一个简单的函数集动态加载,可以帮助开发人员编写可扩展的代码.
- 3.Go插件是使用`go build -buildmode = plugin`构建标志编译
- 4.Go插件包中的导出函数和公开变量,可以使用插件包在运行时查找并绑定调用.
- 5.Go runtime import插件的开发人员必须将插件视为黑盒子,做好各种最坏的假设