Table of Contents

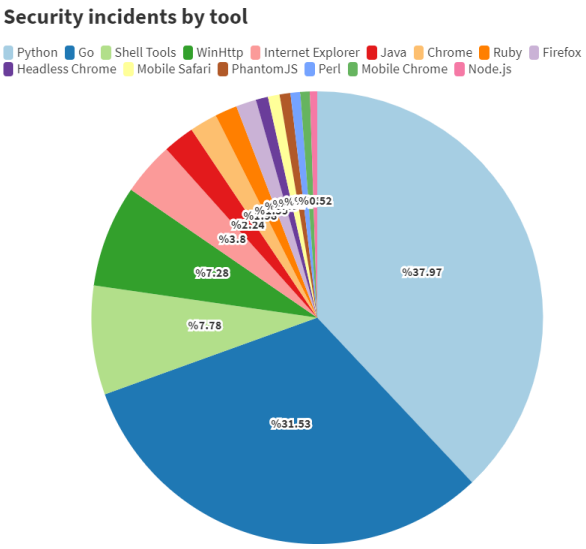
Go语言简介

Go语言于2009年11月正式宣布推出，成为开放源代码项目，并在Linux及Mac OS X平台上进行了实现，后来追加了Windows系统下的实现。

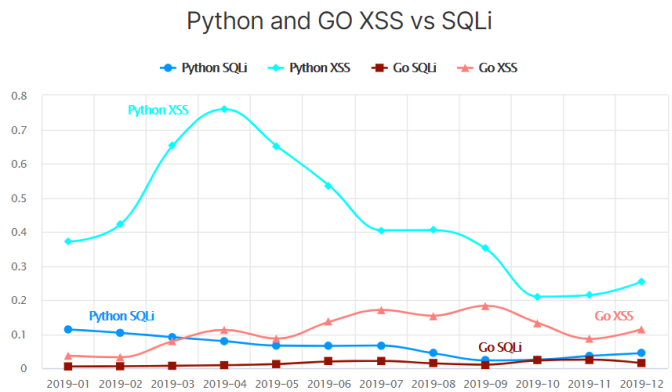
GO具有表现力、简洁、干净、高效。它的并发机制使编写程序更容易，从而最大限度地利用多核和网络化机器，而其新型系统使程序构建更加灵活和模块化。

GO语言黑客编程趋势

网络安全公司 Imperva Cloud WAF 近期分享了其在 2019 年一年内针对网络安全事件的观察。对数据进行聚类分析后，他们得到了如下结论：



按工具来分类，Python 依旧是大多数黑客的首选武器，紧接着是使用量增长迅速的 Go 语言；再其次是 WinHttp 库，该库主要由 Windows 上运行的 .net 和 CPP 使用；Shell 工具（如 cURL、wget 等）也榜上有名。浏览器也是常用的攻击工具。



接下来，该公司又对两种最常见的攻击 ——XSS 和 SQLi—— 以及通过 Go 和 Python 使用这些攻击的尝试进行了观察。值得注意的是，到 2019 年底，Go 在两种类型的攻击中都赶上了 Python。不过，判断这种趋势是否会持续还为时过早，但 Go 变得越来越流行是毋庸置疑的。

助安社区经过简单调研后，国内居然没有一套完整面向网络安全人员go系列课程（可能没有发现，不接受反驳，手动滑稽），为弥补这一遗憾我们尽全力打造出了这一套系列课。

参考：

- Python 和 Go 成为年度最受欢迎的黑客工具榜首

法律声明

请严格遵守《中华人民共和国网络安全法》，本系列课程只可用于合法用途，任何未授权攻击行为都是违法的，由此造成后果与本社区无关，特此声明。

Go语言开发环境安装

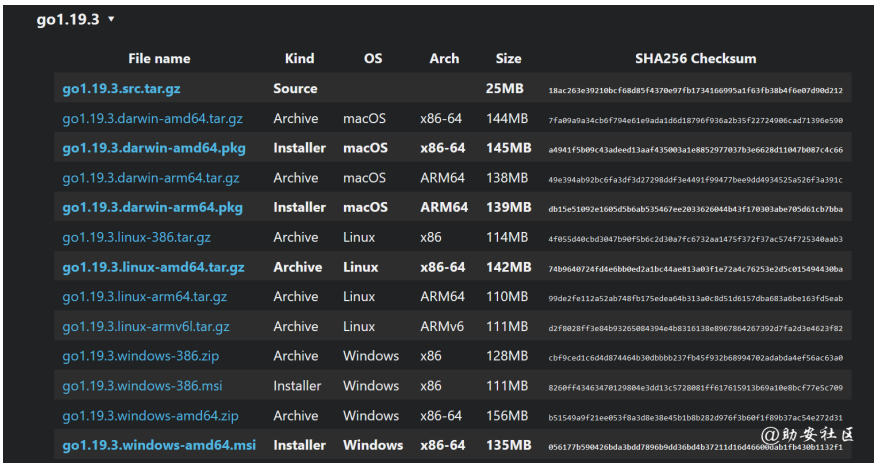
都已经二十一世纪了，开发时再也不是“如毛饮血”的时代，好用的的工具排长队给大家用，这里介绍几款好用Golang集成开发工具：

- Goland（课程里用的就是这个）
- Visual Studio Code
- LiteIDE

安装Go

从官网下载本次教学所使用版本go1.19.3 ，最好一致否则出现问题不知道如何解决。下载地址：

```
官网：https://go.dev/dl/
百度网盘：https://pan.baidu.com/s/19cZK-AVUXE6NtJaphymRow?pw
```



go1.19.3

File name	Kind	OS	Arch	Size	SHA256 Checksum
go1.19.3.src.tar.gz	Source			25MB	18ac263e39210bcf68d85f4370e97fb1734166995a1f63fb38b4f6e07d90d212
go1.19.3.darwin-amd64.tar.gz	Archive	macOS	x86-64	144MB	7fa09a9a34cb6f794e61e9ada1d6d18796f936a2b35f22724906cad71396e590
go1.19.3.darwin-amd64.pkg	Installer	macOS	x86-64	145MB	a4941f5b09c43adeed13aaf435003a1e885297037b3e6628d11047b087c4c66
go1.19.3.darwin-arm64.tar.gz	Archive	macOS	ARM64	138MB	49e394ab92bc6fa3df3d27298ddf3e4491f99477bee9dd4934525a526f3a391c
go1.19.3.darwin-arm64.pkg	Installer	macOS	ARM64	139MB	db15e51092e1605d5b6ab53546 7ee2033626044b43f170303abe705d61cb7bba
go1.19.3.linux-386.tar.gz	Archive	Linux	x86	114MB	4f055d40cbd3047b90f5b6c2d30a7fc6732aa1475f372f37ac574f725340aab3
go1.19.3.linux-amd64.tar.gz	Archive	Linux	x86-64	142MB	74b9640724fd4e6bb0ed2a1bc44ae813a03f1e72a4c76253e2d5c015494430ba
go1.19.3.linux-arm64.tar.gz	Archive	Linux	ARM64	110MB	99de2fe112a52ab748fb175edea64b313a0c8d51d6157dba683a6be163fd5eab
go1.19.3.linux-armv6l.tar.gz	Archive	Linux	ARMv6	111MB	d2f8028ff3e84b93265084394e4b831613 8e096786426739 2d7fa2d3e4623f82
go1.19.3.windows-386.zip	Archive	Windows	x86	128MB	cbf9ced1c6d4d874464b30dbbbb237fb45f932b68994702adabda4ef56ac63a0
go1.19.3.windows-386.msi	Installer	Windows	x86	111MB	8260ff4346347012980 4e3dd13c5728081ff617615913b69a10e8bcf77e5c709
go1.19.3.windows-amd64.zip	Archive	Windows	x86-64	156MB	b51549a9f21ee053f8a3d8e38e45b1b8b282d976f3b60f1f89b37ac54e272d31
go1.19.3.windows-amd64.msi	Installer	Windows	x86-64	135MB	056177b590426bda3bdd7896b9dd36bd4b37211d16d46600aab1fb430b1132f1

安装GOland

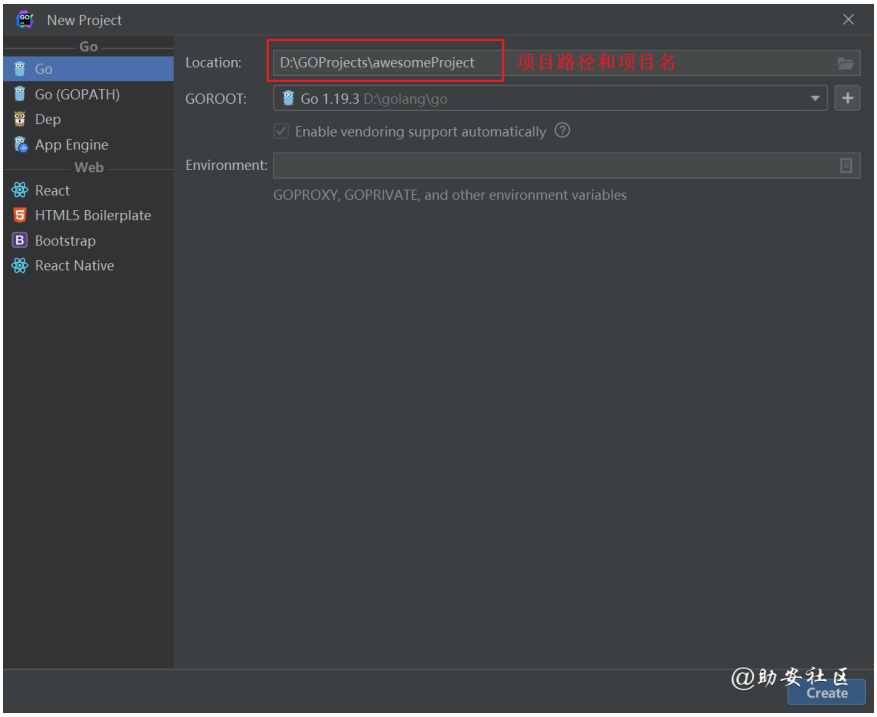
```
官网：https://www.jetbrains.com/go/
百度网盘：https://pan.baidu.com/s/12Ml5hV43F-RxwuRbeq5yxQ?pw
```

Goland可以免费试用30天，继续使用大家可以参考下面链接：

```
https://chenjunan.top/detail/Detail?id=37
```

第一次运行

颤抖吧小伙伴们，这是属于你们的高光时刻，第一个go程序即将诞生。点击 File->New-Project



main.go

```
package main

import "fmt"

func main() {
    fmt.Println("hello world.")
}
```

小结

至此go开发环境已搭建完成，多尝试goland的一些配置，后面遇到问题可以快速解决。

Go语言结构

使用上小节大家写的第一个项目 `awesomeProject` 为例。

项目结构

```
awesomeProject/
|-- .idea
|-- go.mod
`-- main.go
```

上面是最基础的 `GoLand` 创建 `go` 项目目录结构

- .idea 文件夹保存的是当前项目 `GoLand` 配置信息，备份代码时可以删除
- go.mod 有关模块依赖关系信息，导入的模块信息
- *.go `Go` 语言代码存放文件

代码结构

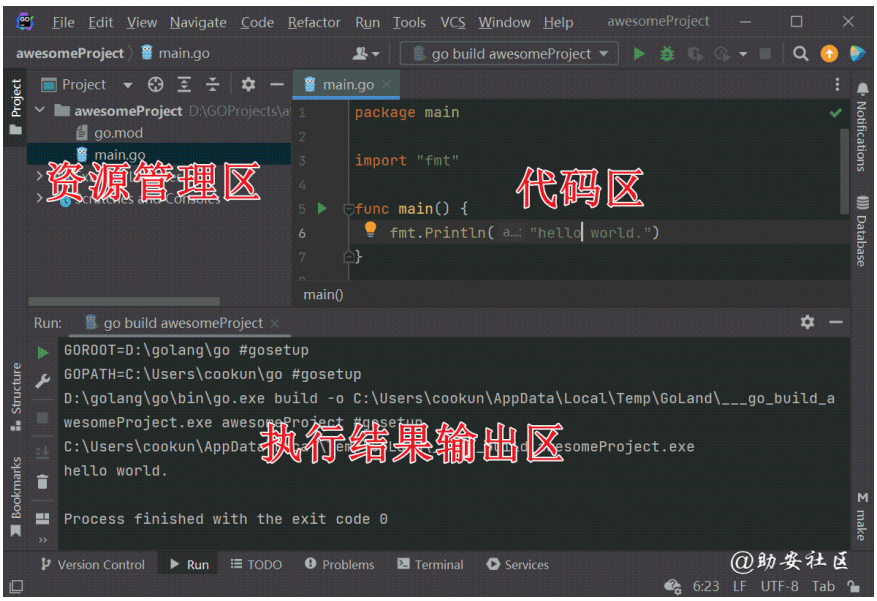
```go
package main

import "fmt"

func main() {
    /*打印*/
    //打印
    fmt.Println("hello world.")
}
```

上面代码里组成有以下几个部分：

- 包声明
- 导入包
- 函数
- 变量
- 语句 & 表达式
- 注释

GoLand

`GoLand` 常用几个区域，资源管理区用于管理文件创建删除，代码区用于编写代码，执行结果输出区可以看到程序执行信息，包括Debug的一些信息。



参考

- Go 语言结构 | 菜鸟教程

Go语言变量

Go 语言变量名由字母、数字、下划线组成，其中**首个字符不能为**数字。

```
var year = 2022
var age, name = 10, "小明"
fmt.Printf("%d 年%s %d 岁", year, name, age)
```

变量类型

变量就像一个桶，里面不光可以装水还能装沙子、石头和冰块等。定义不同变量类型是为了把数据分成所需内存大小，编程时准确使用类型可以充分利用内存和语言特性。

Go 语言按类别有以下几种数据类型：

- 基础类型
  - 布尔型
  - 数字型
  - 字符串型
- 派生类型
  - 指针类型
  - 数组类型
  - 结构化类型
  - Channel 类型
  - 函数类型
  - 切片类型
  - 接口类型
  - Map类型

声明&赋值

看到上面这么多类型是不是发怵，感觉着我能学会吗？这得学什么时候去？莫慌黑客编程初期能运行得到结果就行，毕竟不是上纲上线的项目，没什么效率的担忧，以后技术提升上去再迭代版本。

布尔类型

布尔型的值只可以是 `true` 或者 `false`

```
var a = true
var b = false
```

简化上面赋值过程

```
a := true
b := false
```

也可以先 `声明` 变量，不赋值，要指明变量类型。

```
var a bool // 默认是 false
a = true // 赋值
```

数字类型

数字类型大体分为int、float两种，这里你记住使用 `阿拉伯数字` 等于 `数字类型` 就可以，工具的编写很少用到float（反正我从来没用过）。

```
var a = 1
var b = 65535
```

简化上面赋值过程

```
a := 1
b := 65535
```

也可以先 `声明` 变量，不赋值，要指明变量类型。

```
var a int // 默认是 0
a = 1 // 赋值
```

字符串类型

字符串类型是以后编程中用到最多类型之一。

```
var a = "助安社区"
var b = "君师"
```

简化上面赋值过程

```
a := "助安社区"
b := "君师"
```

也可以先 `声明` 变量，不赋值，要指明变量类型。

```
var a string // 默认是 "" -> 空字符串，不是nil
a = "助安社区" // 赋值
```

数组类型

数组是具有相同唯一类型的一组已编号且长度固定的数据项序列，这种类型可以是任意的原始类型，例如整型、字符串或者自定义类型。


```
var a = [9]int{} // 固定长度数组初始化空数组，数组类型的默认值
var b = [4]bool{true, true} // 赋值前2个索引数据，其余7个为默认值
var c = [...]int{1, 2, 3, 101, 102} // 不定长数组初始化赋值，声
```

访问数组

```
var a = [9]int{1, 2, 3, 4, 5, 6, 7, 8, 9}
fmt.Println(a[3]) // 输出4，切记数组的下标是0
a[0] = 101        // 数组内容赋值
```

切片类型

切片类型类似 `python` 里的列表类型，切片是对数组的抽象。Go 数组的长度不可改变，在特定场景中这样的集合就不太适用，Go 中提供了一种灵活，功能强悍的内置类型切片("动态数组")，与数组相比切片的长度是不固定的，可以追加元素，在追加时可能使切片的容量增大。

```
var a = make([]string, 3) // 初始化， make([]string, 3) 数据的
var b = []int{1, 2, 3, 4} // 初始化并且赋值
```

简化上面赋值过程

```
a := make([]string, 3) // 初始化，数量可以设置为0，切片是可扩展的
b := []int{1, 2, 3, 4} // 初始化并且赋值
```

也可以先 `声明` 变量，不赋值，要指明变量类型。

```
var a []string // 默认是 []
```

切片类型的赋值比基础类型要复杂一些

切片新增数据

下面是经常使用的的方式，初始化一个切片以便循环存储数据。

```
var a = make([]int, 0)
a = append(a, 1)
a = append(a, 2)
```

切片复制数据

```
var a = []int{1, 2, 3}
var b = make([]int, 3) // 这里必须要初始化切片，并且长度要大于等于
copy(b, a)
```

切片长度

使用 len() 方法获取当前长度，cap() 返回指定类型的容量，根据不同类型，返回意义不同。

```
var a = make([]int, 3, 9)
a = append(a, 1)
a = append(a, 2)
fmt.Println(len(a), cap(a))
```

切片截取

使用[索引]读取切片里的一条数据，使用[开始索引:结束索引]读取多条数据，读取出“一小块”连续数据。

```
var a = []int{1, 2, 3, 4, 5, 6, 7, 8, 9}
fmt.Println(a[0])    // 读取出索引0里的数据 1，输出：1
fmt.Println(a[:3])   // 读取索引0(包含) 到索引3(不包含)的数据，输
fmt.Println(a[4:])   // 读取索引4(包含) 到索引len(a)的数据，输出
fmt.Println(a[4:6]) // 读取索引4(包含) 到索引6(不包含)的数据，输
// 从0开始，左包含右不包含
```

指针类型

学习c语言的时候指针就是噩梦，不过Go 语言中指针是很容易学习的。

```
var a = "助安社区"
var b *string // 声明一个字符串指针
b = &a        // 指针赋值，&符号的意思代表取后面变量地址
```

简化上面赋值过程

```
var a = "助安社区"
b := &a
```

如何赋值大家已经知道了，但可能有人不懂指针是什么，这里简单科普下：


```
var a = "助安社区"
var b = 101
var c = 999
var ap = &a
var bp = &b
var cp = &c
```

使用指针里面的值，思考下面代码中变量 `b` 的值是多少？

```
var b = 101
var bp = &b // & 取变量的地址
*bp = 102   // * 是取出指针地址里的值
```

结构体类型

Go 语言中数组可以存储同一类型的数据，在结构体中则可以定义不同数据类型。结构体一般用于具有相同类型或不同类型的数据构成的数据集合。例如要定义一个人的结构体：

```
type people struct {
    name     string // 名字
    age      int    // 年龄
    birthday string // 生日
    gender   string // 性别
}
```

定义结构提变量，初始化赋值

```
// 定义时直接赋值
var a = people{
    name:     "君师",
    age:      30,
    birthday: "10/10",
    gender:   "男",
}

// 先定义后赋值
var b = people{}
b.name = "君师"
b.age = 33
b.birthday = "10/10"
b.gender = "男"
```

Map类型(集合)

Map 是一种无序的键值对的集合，通过 key 来快速检索数据。

```
// 声明map集合变量
var a map[string]string
// 创建集合，前面只是声明并没有创建
a = make(map[string]string)
// 集合赋值
a["name"] = "君师"
a["birthday"] = "10/10"
a["gender"] = "男"

// 声明变量并且赋值
var b = map[string]string{
    "name":     "君师",
    "birthday": "10/10",
    "gender":   "男",
}
```

看到这里大家是否想到一个熟悉的数据 `json` ，没错编写工具很多时候都是将 `json` 转为 `map` 集合使用。

```
{
    "name":     "君师",
    "birthday": "10/10",
    "gender":   "男",
}
```