出题人wp

出这道题的背景是出题人在学go，就像让大家都~~折磨一下~~玩一玩

众所周知，go逆向是块硬骨头，主要问题在于：通常难以定位主要逻辑进行分析

考虑到本次比赛主要受众是新生，所以没有扣符号表，降低了难度

题目所给的二进制文件是个http server，运行后会在本机8080端口开启服务，监听本机（必须是127.0.0.1）特定http请求进行响应（需要了解一些Get、Post方法的相关知识），题目源码已开源，github地址：https://github.com/DX39061/MoeCTF2022_gogogo

但其实要做这道题仅仅静态分析就足够了，低版本IDA可以使用IDA Golang Helper优化伪代码，出题人的IDA7.7自动分析就已经很好看了。对于go逆向，当你没啥思路时，建议从后往前翻函数列表（算是经验吧

处理flag相关http请求的逻辑在`main_flagHandler`（地址0x7432A0，基址为0，下同）

```
void __fastcall main_flagHandler()
{
  __int64 v0; // rax
  __int64 v1; // r14
  __int128 v2; // xmm15
  char *v3; // rax
  _QWORD *v4; // rax
  _QWORD *v5; // rax
  _QWORD *v6; // rax
  __int64 v7; // [rsp-20h] [rbp-50h]
  __int64 v8; // [rsp-20h] [rbp-50h]
  void *retaddr; // [rsp+30h] [rbp+0h] BYREF
  __int64 v10; // [rsp+38h] [rbp+8h]

  if ( (unsigned __int64)&retaddr <= *(_QWORD *)(v1 + 16) )
    runtime_morestack_noctxt_abi0();
  v10 = v0;
  v3 = (char *)github_com_gin_gonic_gin__ptr_Context_Query();
  qword_AD3898 = (__int64)"flagfnofform";
  if ( runtime_writeBarrier )
    runtime_gcWriteBarrier();
  else
    main_flag = v3;
  if ( !"flagfnofform" )
  {
    runtime_makemap_small();
    v7 = runtime_mapassign_faststr();
    *v4 = &unk_771820;
    if ( !runtime_writeBarrier )
    {
      v4[1] = &off_89A170;
LABEL_9:
      runtime_convT();
      github_com_gin_gonic_gin__ptr_Context_Render(v7);
      *(_BYTE *)(v10 + 104) = 63;
      return;
```

```
    }
LABEL_8:
    runtime_gcWriteBarrierCX();
    goto LABEL_9;
  }
  if ( !(unsigned __int8)main_check() )
  {
    runtime_makemap_small();
    v7 = runtime_mapassign_faststr();
    *v6 = &unk_771820;
    if ( !runtime_writeBarrier )
    {
      v6[1] = &off_89A180;
      goto LABEL_9;
    }
    goto LABEL_8;
  }
  runtime_makemap_small();
  v8 = runtime_mapassign_faststr();
  *v5 = &unk_771820;
  if ( runtime_writeBarrier )
    runtime_gcWriteBarrierCX();
  else
    v5[1] = &off_89A100;
  runtime_convT();
  github_com_gin_gonic_gin__ptr_Context_Render(v8);
  *(_OWORD *)(v10 + 216) = v2;
  if ( runtime_writeBarrier )
    runtime_gcWriteBarrier();
  else
    *(_QWORD *)(v10 + 208) = 0LL;
}
```

主要逻辑在`main_check`（地址0x7423E0）

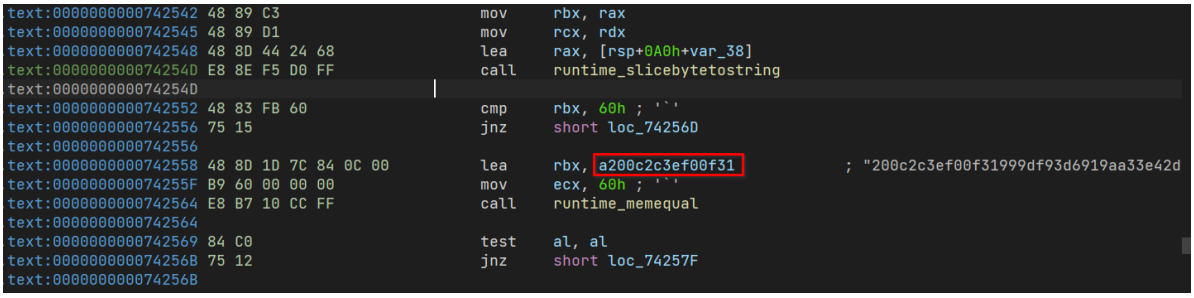
```
_BOOL8 __fastcall main_check()
{
  __int64 v0; // rax
  __int64 v1; // r14
  __int64 v2; // rax
  __int64 v3; // rax
  unsigned __int64 v4; // rdx
  __int64 v5; // rsi
  __int64 v6; // rbx
  unsigned __int64 v7; // rdi
  unsigned __int8 v9; // r9
  char v10; // r9
  __int64 v11; // rbx
  __int64 v12[2]; // [rsp+10h] [rbp-48h] BYREF
  char v13; // [rsp+38h] [rbp-20h] BYREF
  __int64 v14; // [rsp+40h] [rbp-18h]
  void *v15; // [rsp+48h] [rbp-10h]
  __int64 v16; // [rsp+60h] [rbp+8h]

  if ( (unsigned __int64)&v13 <= *(_QWORD *)(v1 + 16) )
```

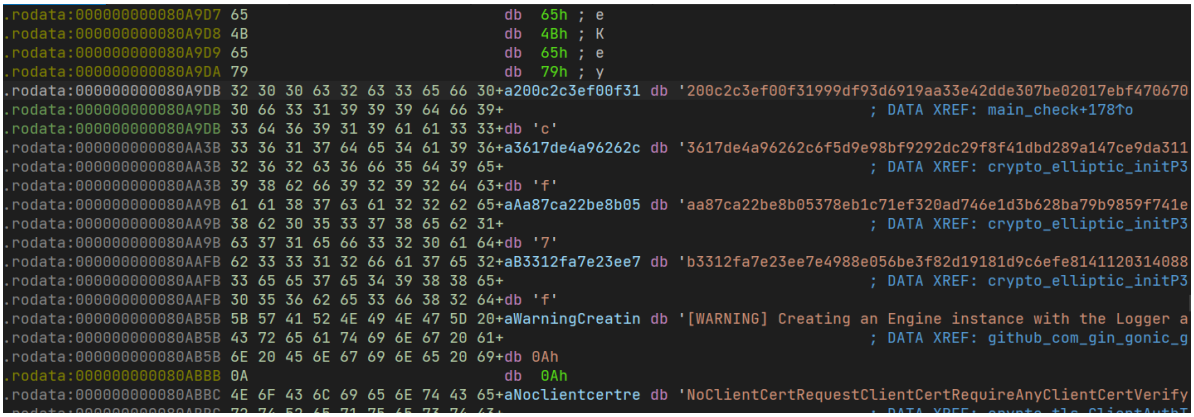
```
    runtime_morestack_noctxt_abi0();
  v16 = v0;
  qmemcpy(v12, "---moeCTF2022---", sizeof(v12));
  v15 = (void *)runtime_newobject();
  qmemcpy(v15, "---moeCTF2022---", 16);
  runtime_stringtoslicebyte();
  v2 = main_AesEncrypt();
  if ( v12 )
    return 0LL;
  v14 = v2;
  v3 = runtime_makeslice();
  v4 = 2 * v16;
  v5 = v14;
  v6 = 0LL;
  v7 = 0LL;
  while ( v16 > v6 )
  {
    v9 = *(_BYTE *)(v6 + v5);
    if ( v7 >= v4 )
      runtime_panicIndex();
    *(_BYTE *)(v3 + v7) = a0123456789abcd_0[v9 >> 4];
    v10 = a0123456789abcd_0[v9 & 0xF];
    if ( v4 <= v7 + 1 )
      runtime_panicIndex();
    *(_BYTE *)(v7 + v3 + 1) = v10;
    ++v6;
    v7 += 2LL;
  }
  v11 = v3;
  runtime_slicebytetostring();
  return v11 == 96 && (unsigned __int8)runtime_memequal();
}
```

实际就是对flag进行AES加密（模式为CBC，key和iv都是`---moeCTF2022---`），与密文比对，源代码如下：

```
func check(flag string) bool {
    encFlag := "200c2c3ef00f31999df93d6919aa33e42dde307be02017ebf47067099ed0bddc525d5dba0f83c122159b89ae715907cc"
    key := []byte("---moeCTF2022---")
    iv := []byte("---moeCTF2022---")
    encrypt, err := AesEncrypt([]byte(flag), key, iv)
    if err != nil {
        return false
    }
    if hex.EncodeToString(encrypt) == encFlag {
        return true
    }
    return false
}
```

密文需查看汇编代码才能找到





由于go存储字符串末尾没有`\0`，所以某些版本IDA可能会无法分割上面这些数据，混在一起。这时可以由上面伪代码倒数第二行`v11 == 96`猜出密文长度为96位，从而手动分割数据拿到密文

至于密文解密，出题时候亲测在线网站可解，网上各种脚本也很多，就不赘述

flag：`moeCTF{g0l@ng_1s_4n_1nte^est1n9_lan9ua9e}`

大佬wp

> 描述:RX神轻易地攻破了DX的服务器，拿到了服务端程序，随便看了一眼，说这道题不是有手就行?

运行一下，让你访问http://localhost:8080/welcome然后显示消息: "Welcome to MoeCTF2022, can you find the flag?"

肯定是找不到的,ida打开看一看,没去符号表,稍微调了一下,发现用到了很多`github_com_gin`开头的函数,查了一下,是go的一个web框架GO语言GIN框架入门

将上文阅读了一点后,就能知道在访问http://localhost:8080/welcome时可能是`main.welcomeHandler`处理了GET请求,显示消息,源码可能如下:

```
package main

import "github.com/gin-gonic/gin"

func main() {
    // 创建一个默认的路由引擎
    engine := gin.Default()
    // GET：请求方式；/welcome：请求的路径
    // 当客户端以GET方法请求/welcome路径时，会执行后面的匿名函数
    engine.GET("/welcome", func(context *gin.Context) {
        //返回JSON格式的数据
        context.JSON(200, gin.H{
            "message": "Welcome to MoeCTF2022, can you find the flag?",
```

```
        })
    })
    // 启动HTTP服务，默认在0.0.0.0:8080启动服务
    engine.Run()
}
```

而`main.findHandler`,`main.flagHandler`应该也类似,都是对请求进行处理

试了试访问http://localhost:8080/find,结果是 "password required"

调试了一会,发现请求/find接口的时候会先跑到`main_authRequired`处,取出GET请求的参数,和==---moeCTF2022---==对比,于是访问http://localhost:8080/find?password=---moeCTF2022---,结果是 "you are so close to get flag"

试了试http://localhost:8080/flag ,404了,调了好一会也不知道怎么进到`flagHandler`,查着查着试了试http://localhost:8080/find/flag?password=---moeCTF2022---,显示 "please input your flag and I will check it",成功进到`flagHandler`了

交叉引用发现函数表里的`main_check`,`main_AESEncrypt`,`main_PKCS5Padding`都和`flagHandler`有关

查资料的时候发现golang中crypto/aes包 - 简书 (jianshu.com) 调试过程中经过的函数和这个差不多:

```
// 加密
func encryptAES(src []byte, key []byte) ([]byte, error) {
    block, err := aes.NewCipher(key)
    if err != nil {
        return nil, err
    }
    src = padding(src, block.BlockSize())
    blockMode := cipher.NewCBCEncrypter(block, key)
    blockMode.CryptBlocks(src, src)
    return src, nil
}
```

总结下来就是设立16位的==密钥 ---moeCTF2022---==,==PKCS5填充==输入的flag,选取==AES-128, CBC加密模式==,==偏移量 ---moeCTF2022---==

值得注意的是,ida并没有把`main_check`最后对比的参数反汇编出来,需要看汇编才能找到参数:

```
200c2c3ef00f31999df93d6919aa33e42dde307be02017ebf47067099ed0bddc525d5dba0f83c122
159b89ae715907cc
```

拿去在线网站解一下就出了

> moeCTF{g0l@ng_1s_4n_1nte^est1n9_lan9ua9e}