# 自己动手编写远控工具及检测思路

tammypi FreeBuf 今天

PS：本文仅用于技术讨论与交流，严禁用于任何非法用途，违者后果自负

在学习攻击渗透的过程中，不免会接触远控工具。远控工具一般包含服务端和客户端，服务端运行在攻击者的VPS主机上，客户端运行在被攻击机器上。服务端向客户端发送指令，客户端执行指令并将结果回传给服务端，从而达到通过网络远程控制被攻击主机的效果。

现有的远控工具很多，从大名鼎鼎的冰河到CHAOS。但是直接使用现有的远控工具，一方面会担心工具被人加入了后门在运行的过程中自己反而成了被控制方，另一方面只会使用工具也会沦为“脚本小子”而不知道其背后的原理。

本文详细介绍了一款最小功能集的远控工具的实现细节，按照步骤动手实现，不仅可以对于远控工具的背后思路有了更深的体会，也可以巩固自己在多线程、网络编程等方面的知识。

## 一、总体结构

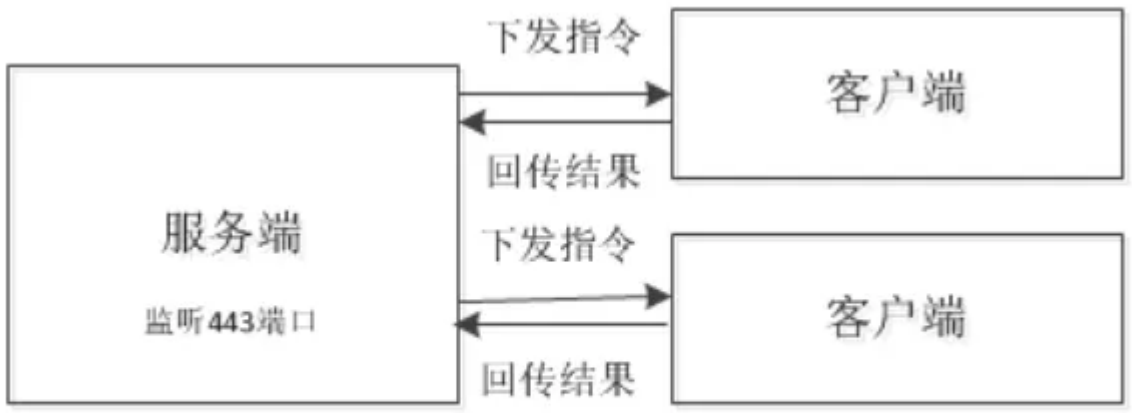


图1. 总体结构

同一般的远控工具一致，我们要实现的这款工具也是包含服务端和客户端。

> 服务端运行在VPS主机上，监听443端口。由于443为HTTPS协议端口，这样被攻击主机上即使监控到存在外联443端口的流量，也不太容易引起重视。所以在模拟攻击者的过程中，可以选用一些常用端口作为服务端的监听端口。
>
> 服务端和客户端使用TCP协议进行通信，利用多线程服务端处理多个客户端同时在线的情况。
>
> 客户端捕获异常，当与服务端的通信连接断开触发异常时，延时5秒后重新发送连接请求。这样即使服务端出现了中途退出的情况，重新运行后客户端也可以重新上线。

## 二、实现细节

### 2.1 服务端

服务端主要做两件事情：

> 使用socket监听443端口，使用独立的线程完成与客户端的通信，对于指定客户端下发指令并打印指令执行结果
>
> 在主线程中死循环监听攻击指令输入（如snapshot对于被攻击机进行截屏、cmd在被攻击机上执行系统命令）

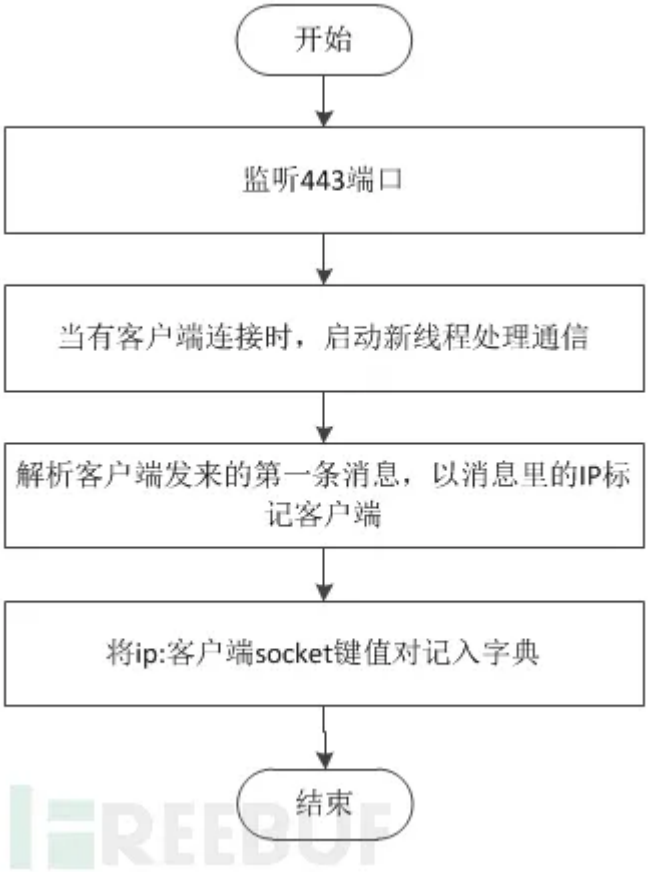


图2.服务端记录客户端连接流程

图2即是服务端记录客户端连接的流程。客户端发来的第一条消息会是“HELO,客户端IP”，服务端会解析第一条消息，并以客户端IP为key将客户端的socket保存到字典里。后续，当需要与指定的客户端进行通信时，直接从字典里根据IP拿到对应客户端的socket进行消息的发送和接收即可。

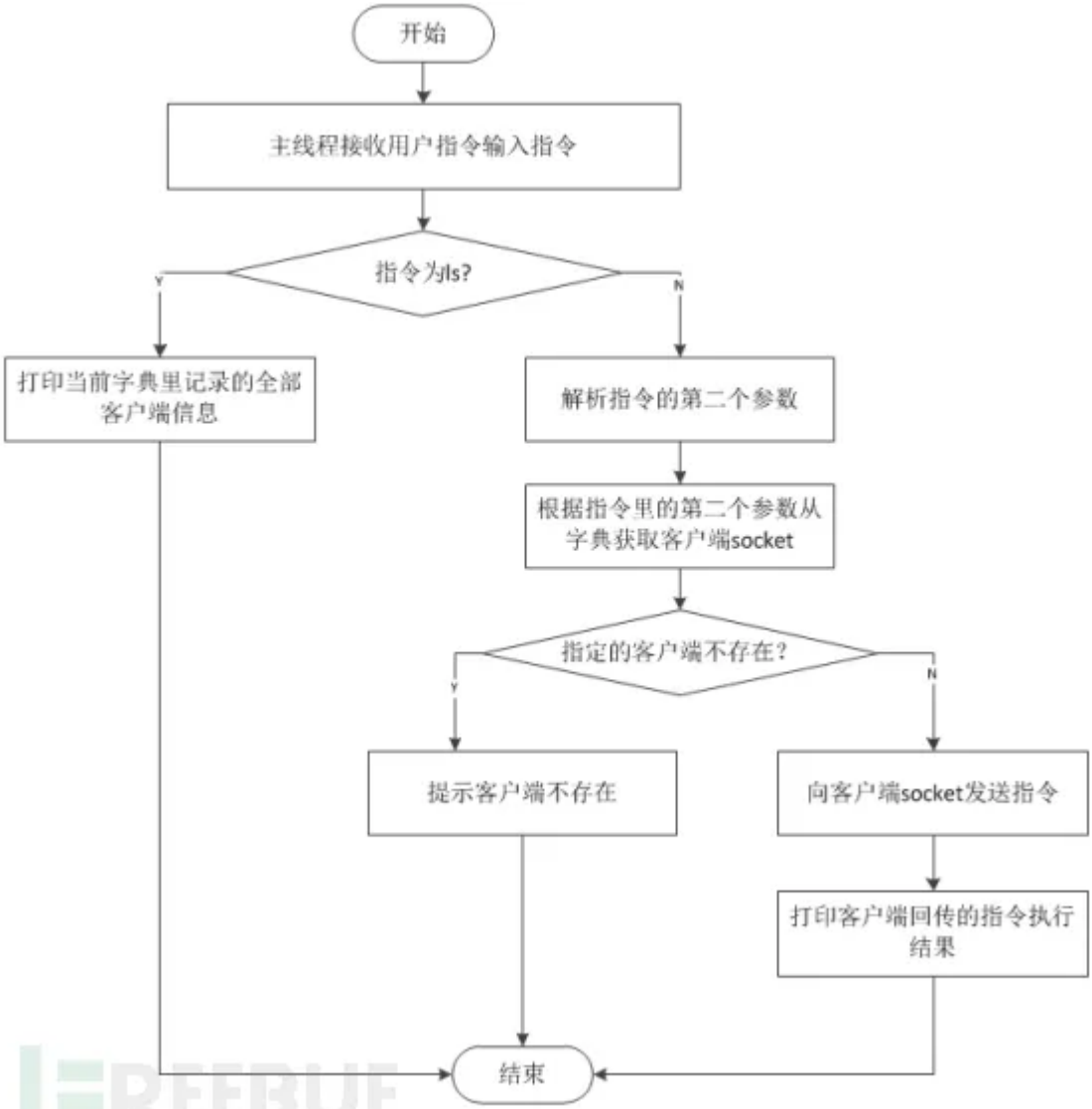


图3.服务端处理指令流程

图3则是服务端处理指令的流程。目前服务端接收的指令如下：

> ls: 打印当前全部客户端IP
>
> snapshot: 对于指定客户端所在机器进行截屏。示例：snapshot ${client_ip}
>
> clipboard: 获取客户端所在机器的剪贴板内容。示例：clipboard ${client_ip}
>
> cmd: 在指定客户端所在机器执行系统命令。示例：cmd ${client_ip} ${command}

> showdriver:展示客户端所在机器的全部盘符。示例：showdriver ${client_ip}
>
> getfilelist:获取客户端所在机器指定路径下的文件列表。示例：getfilelist ${client_ip} ${path}
>
> getfile:获取客户端所在机器指定文件。示例：getfile ${client_ip} ${filepath}

当服务端接收ls指令时，直接将记录有客户端socket字典的key列表打印出来即可。当收到的是其他指令时，由于指令的第二个参数是客户端IP，那么根据IP得到对应的客户端socket，使用socket将指令发送给客户端，并打印客户端socket的返回结果即可。

## 2.2 客户端

客户端主动连接服务端的443端口。在初次连接上时，主动发送一条“HELO,客户端IP”消息标记自己。后续客户端则是持续循环等待服务端发送过来的指令，并根据指令真实的在被攻击上进行截图、获取文件列表等操作，并将结果回传给服务端。

客户端会捕获异常，当socket出现异常后，等待5秒尝试重连。

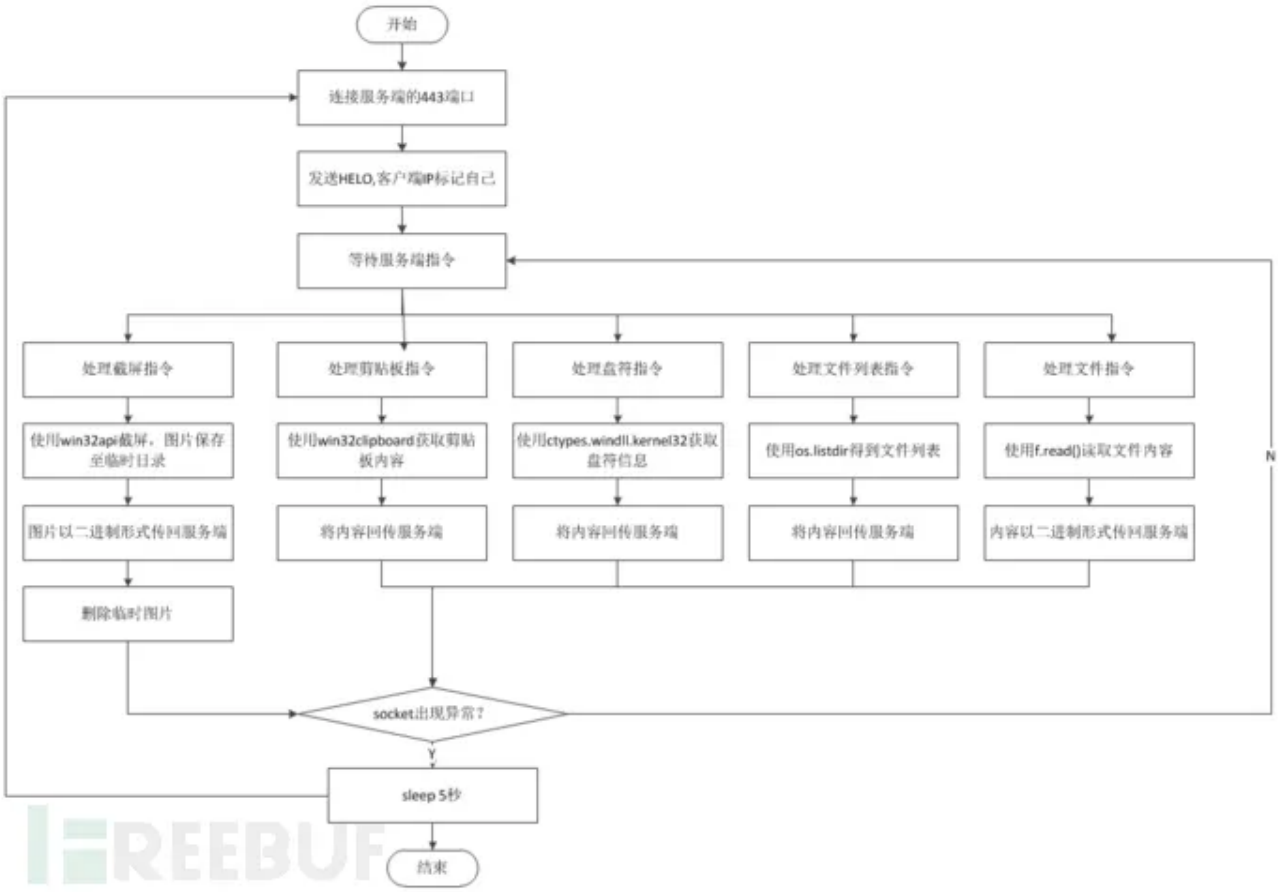

图4. 客户端处理指令流程

## 三、重点问题解决

### 3.1持久化

一个远控工具除了支持服务端下发指令，客户端执行指令并回传结果的功能外，还需要可以对于自己进行持久化。所谓持久化，即保证客户端进程被杀死或者被攻击主机关机后，也可以在被攻击主机再次开机时启动。

进行持久化有如下几种方法：

> 写注册表
>
> 将执行文件复制到启动目录
>
> 将自己注入其他系统进程

这几种方法各有优缺点，将执行文件复制到启动目录一般需要管理员权限，在执行客户端进程的用户没有管理员权限时，复制会导致“Permission denied”错误。而写注册表和注入进程的行为，通常会被杀软作为高危项检测到。

由于我们的这款远控工具只是学习使用，所以不考虑绕过杀软的方法，使用写注册表的方式将自己设置为开机自启动：

```
ROOTPATH = os.getcwd()
EXEPATH = ROOTPATH + "/" + "video.exe"
try:
  temp_filepath = os.path.normpath(tempfile.gettempdir() + "/svhost.exe")
  shutil.copy(EXEPATH, temp_filepath)
  runpath = "Software\Microsoft\Windows\CurrentVersion\Run"
  hKey = win32api.RegOpenKeyEx(win32con.HKEY_CURRENT_USER, runpath, 0, win32con.KEY_ALL_ACCESS)
  win32api.RegSetValueEx(hKey, "MyTool", 0, win32con.REG_SZ, temp_filepath)
except:
  pass
```

图5. 写注册表开机自启动

主要的步骤就是首先将自己复制到临时目录，并且改名为svhost.exe(与系统进程svchost.exe一字之差)，同时将自己写入注册表的开机自启动项。

### 3.2 TCP Socket通信粘包问题

由于我们实现服务端和客户端的通信是用的TCP协议，而TCP协议容易出现粘包问题，导致服务端接收到的客户端回传的图片或者文件是损坏状态，无法正确打开。

所谓粘包即客户端发送给服务端的多个包被粘在了一起，它产生的主要原因在于接收到数据包放在了缓冲区里，如果缓冲区包写入的速度大于服务端从缓冲区取的速度，服务端就会取到首尾相连的多个包。

为了让服务端可以正确的区分接收到的数据，解决粘包问题，我们需要将传输的数据进行格式化。

我们定义：

> 一条消息的头部4个字节为消息内容长度
>
> 头部后面再跟消息内容

```
def send_content(sock, content):
    total_size = len(content)
    header = struct.pack("i", total_size)
    sock.send(header)
    sock.send(content)
```

图6. 发送格式化的消息

当服务端接收到数据时，先解析头部4个字节得到消息的长度，再根据这个长度得到消息正文。



图7. 处理格式化的消息

### 3.3 客户端打包为exe

我们的这款远控工具使用的python语言进行开发，而客户端主要运行在windows主机上。由于windows主机默认并没有安装python环境，所以需要将客户端打包为exe使得它可以直接在windows主机上进行运行。

将python打包为exe主要使用pyInstaller这款神器，使用-F参数生成单个执行文件，使用-w参数表示运行时去掉控制台窗口。打包完成后，dist目录下即会出现生成的exe文件。

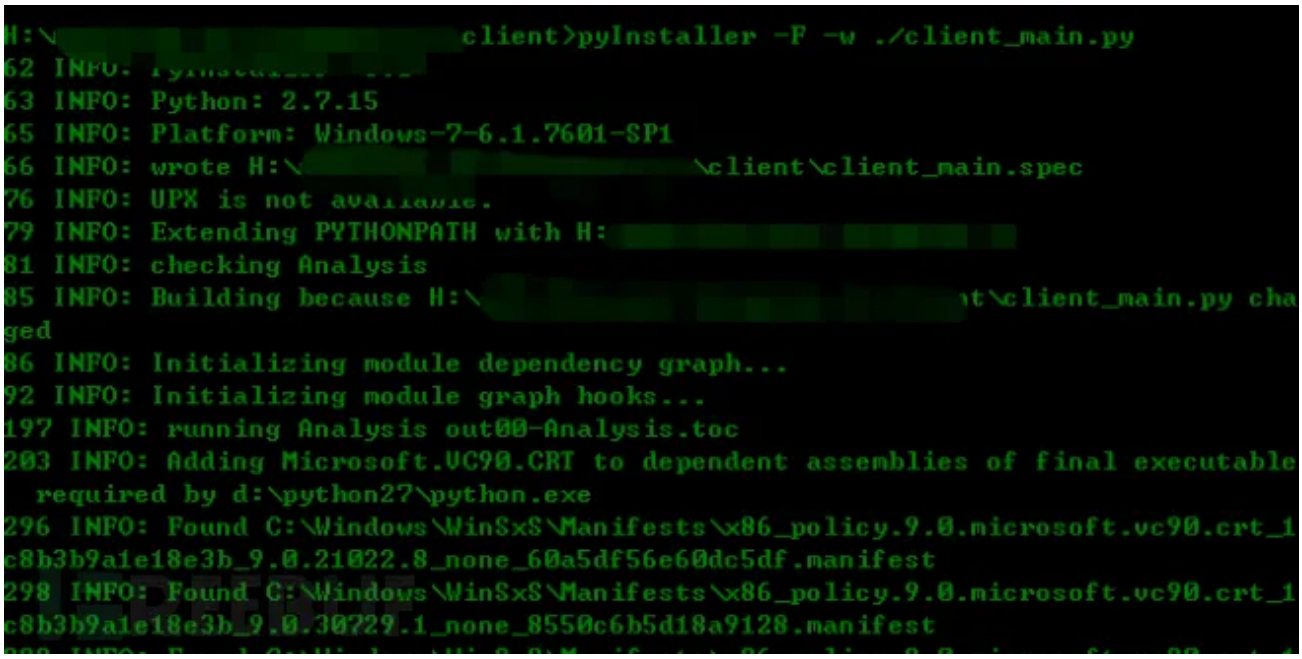


图8. 将客户端打包成exe

## 四、效果演示

服务端运行时，当客户端上线时，会打印客户端的IP。

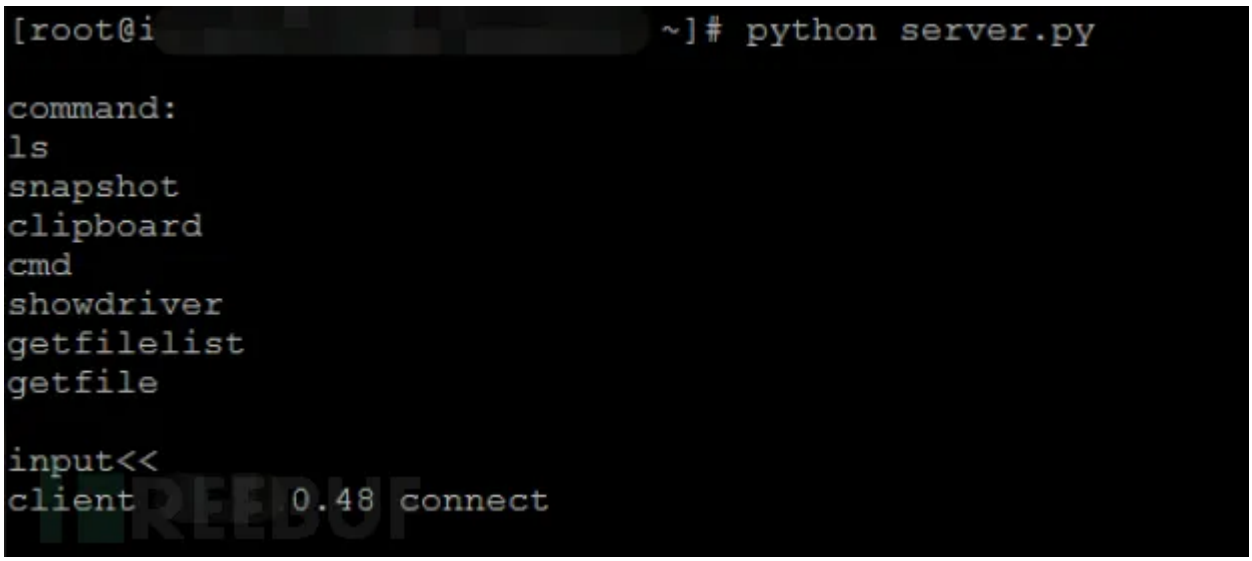

图9.服务端运行效果

使用ls命令，可以打印当前已经上线的全部客户端IP：

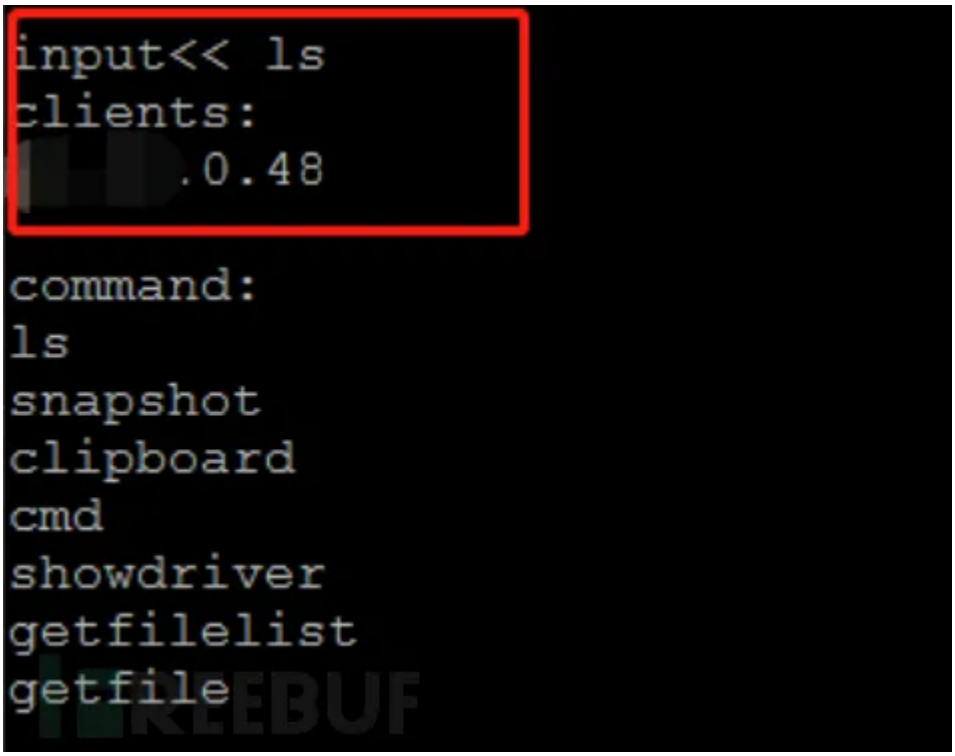


图10.打印全部客户端

可以对于客户端所在机器进行截屏：

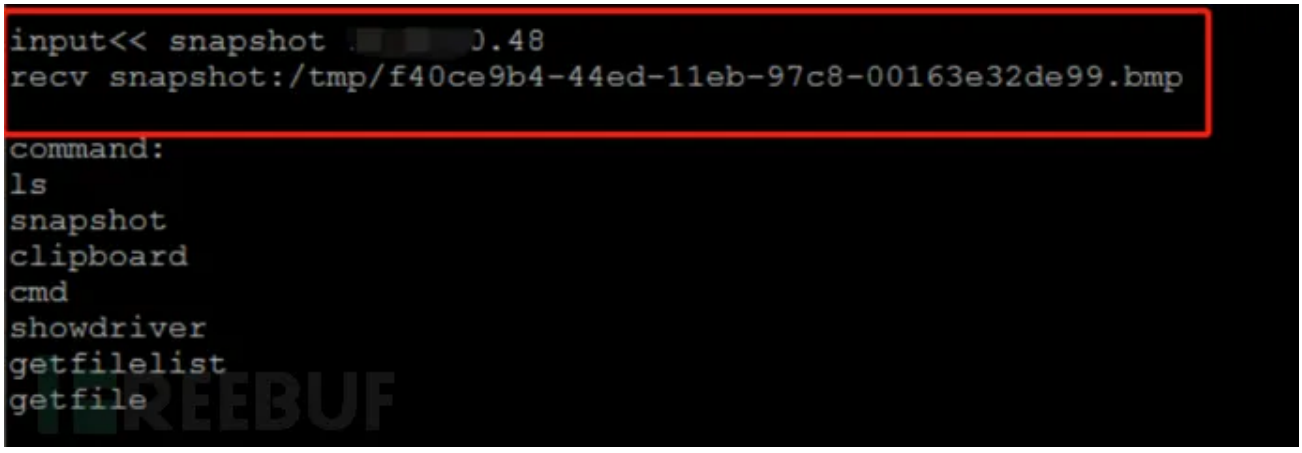


图11.对于客户端所在机器进行截屏

## 五、源码下载地址

https://github.com/tammypi/remote_control_tool

强调：请注意，本源码仅供学习使用，请勿用作非法用途。

## 六、检测思路

在对于这类远控工具进行检测时，重要的特征有以下两点：

1.注册表写入动作、复制自身动作

2.虽然与常用端口进行通信，但是数据包的特征却明显并非为对应协议的特征

阅读原文

喜欢此内容的人还喜欢

再见，360安全卫士!

最码农

---

Webshell免杀的思考与学习

LemonSec

---

彻底揭秘负载均衡算法与实现！深入剖析负载均衡核心

架构之美