软件安全之远程木马

设计与防护

题 目： 远程控制木马设计

组 员：彭馨勇 2017301510012

徐锦盈 2017301510026

范子颖 2017301510008

邓洋涛 2017301510056

课程名称： 软件安全

任课教师： 彭国军

二零一九年六月二十六日

目录

[**一、** **基本描述** 3](#_Toc12391731)

[**二、** **功能实现** 3](#_Toc12391732)

[**2.1 基本架构** 3](#_Toc12391733)

[**2.1.1 用户界面如下：** 3](#_Toc12391734)

[**2.1.2目录层次：** 4](#_Toc12391735)

[**2.1.2.1 server下：** 4](#_Toc12391736)

[**2.1.2.2 client下：** 5](#_Toc12391737)

[**2.2 进程管理实现思路** 6](#_Toc12391738)

[**2.3 进程管理具体实现** 6](#_Toc12391739)

[**2.3.1 构造方法:** 7](#_Toc12391740)

[**2.3.2杀死进程：** 7](#_Toc12391741)

[**2.4 摄像头监控实现思路** 8](#_Toc12391742)

[**2.5 摄像头监控具体实现** 8](#_Toc12391743)

[**2.5.1 引入依赖：** 8](#_Toc12391744)

[**2.5.2 实例化** 9](#_Toc12391745)

[**2.5.3 效果** 9](#_Toc12391746)

[**2.6 屏幕监控实现思路** 9](#_Toc12391747)

[**2.7 屏幕监控具体实现** 10](#_Toc12391748)

[**2.7.1 实例化robot对象** 10](#_Toc12391749)

[**2.7.2 图像编码** 10](#_Toc12391750)

[**2.7.3 效果** 10](#_Toc12391751)

[**三、** **防护软件设计** 11](#_Toc12391752)

1. **基本描述**

本软件基于IDEA开发，采用java语言编写，充分体现了java面向对象和跨平台的优势，具有完整的图形界面，能够与用户进行交互，分为服务端和客户端两部分，基于socket采用正向连接方式传输数据，服务端开启端口进行监听，客户端连接服务端并接收传输过来的数据。

主要功能有：

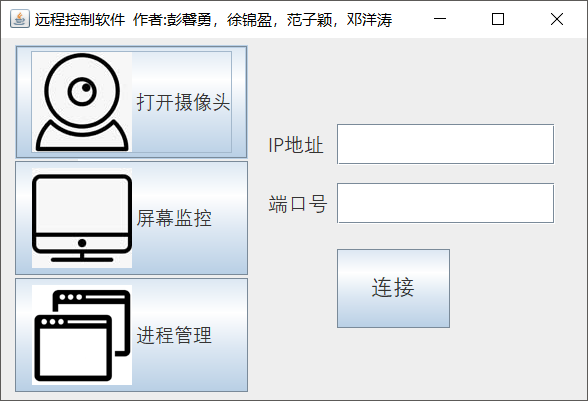
1. 进程管理；
2. 摄像头监控；
3. 屏幕监控。
4. 在启动服务之前有弹窗提示是否允许进行管理，可以很方便地进行远程主机的管理。
5. **功能实现**

**2.1 基本架构**

本软件采用socket进行正向连接的方式，服务端运行开启各种服务并进行端口监听，客户端连接服务端并读取数据。

基本架构分为：用户界面，server端，client端。

**2.1.1 用户界面如下：**



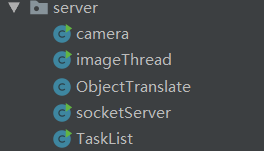
* 连接成功的提示：



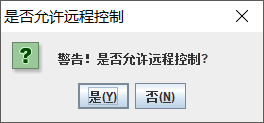
**2.1.2目录层次：**

分为server和client两个文件夹：

**2.1.2.1 server下：**



* socketServer类是服务端的入口。负责开启端口进行监听，在开启端口监听之前有明确的消息提示框如下：

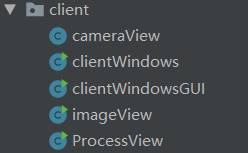


* 当用户选择否的时候，弹出一个警告窗之后退出：

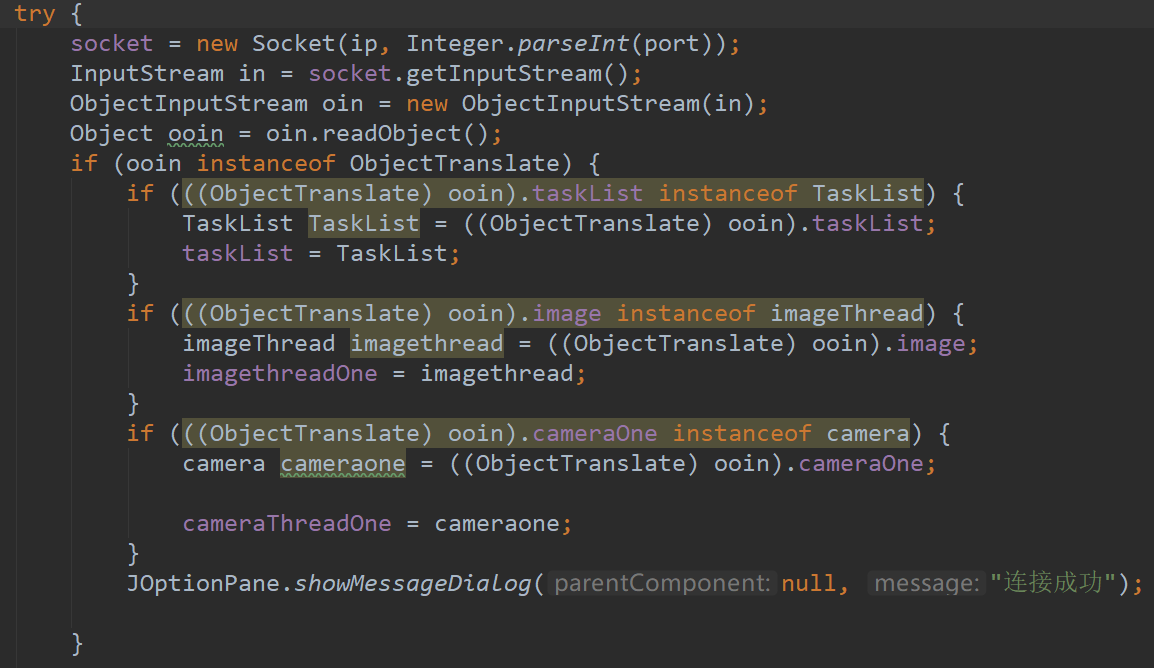


* camera类，imageThread类，TaskList类分别实现了摄像头监控，屏幕监控，进程管理的功能。
* ObjectTranslate类将上述类进行向上转型并封装到一起，经过序列化之后传送到客户端进行反序列化。

**2.1.2.2 client下：**



* clientWindowsGUI是入口类，负责将接收到的数据向下转型同时传送给cameraView类，imageView类，ProcessView类。
* 具体实现如下：

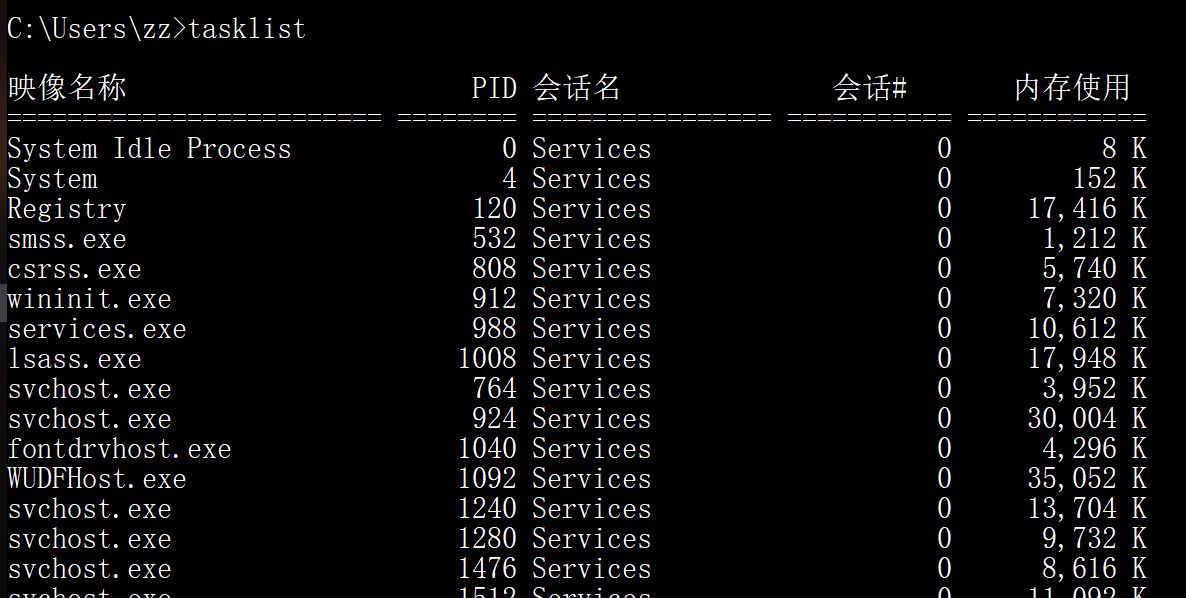


经过反序列化的数据会传送至其他类中进行处理并呈现出来。

**2.2 进程管理实现思路**

* 进程管理实现的思路：

在控制台中输入taskList命令之后会列出当前主机上运行的所有进程，我们只需要执行这一条系统命令即可，同时将数据进行一定的处理呈现出来。杀死进程的思路是通过执行taskkill命令来实现的。



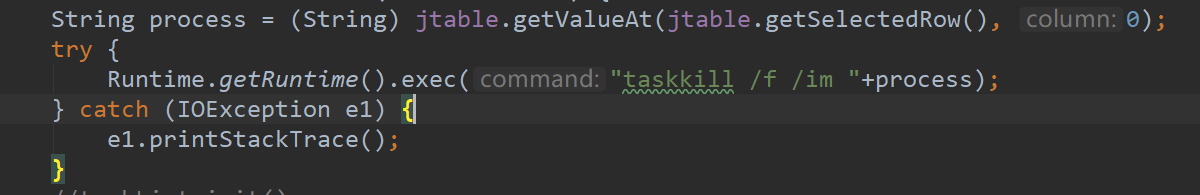
**2.3 进程管理具体实现**

具体实现如下：

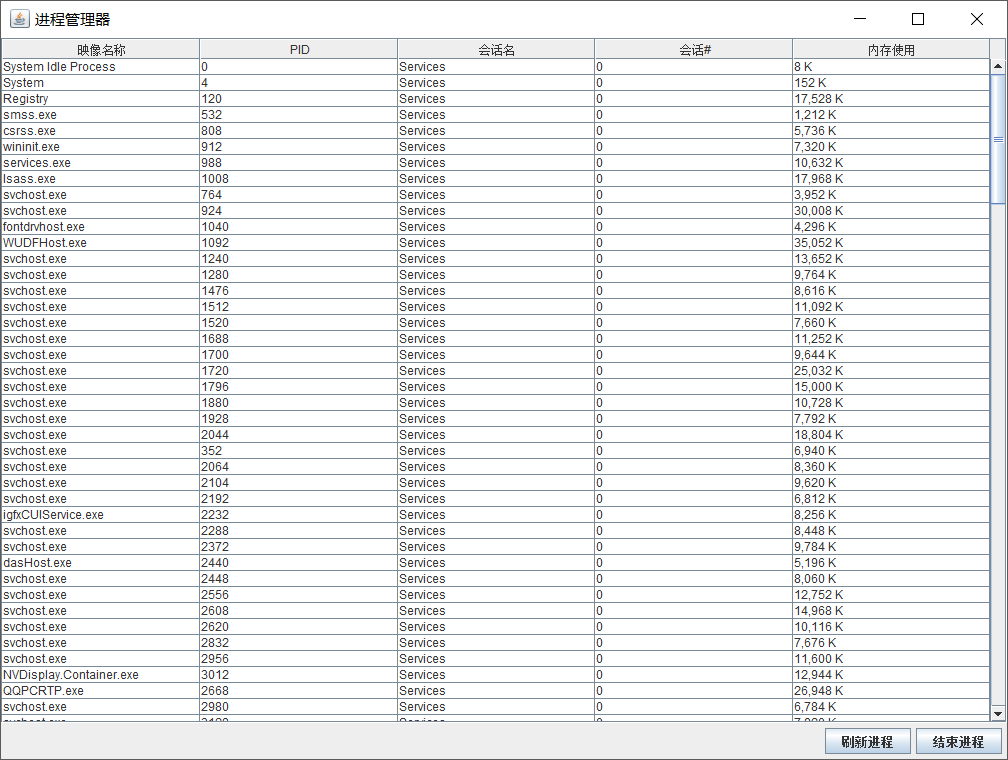
**2.3.1 构造方法:**



**2.3.2杀死进程：**

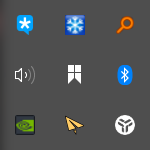


* 管理的界面如下：



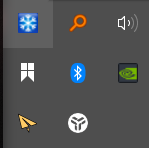
* 杀死进程示例：

TIM.exe在运行





* 杀死TIM.exe进程：



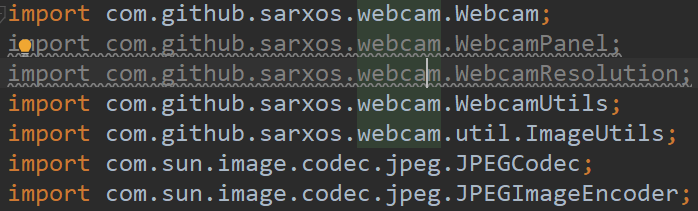
进程已结束。

**2.4 摄像头监控实现思路**

摄像头的调用是通过第三方库webcam-capture来实现的，可以通过Maven包管理工具来引入，可以直接在IDEA中导入。我们将其导入之后实例化对象，然后调用其capture方法来获取调用摄像头之后拍的照片。然后将图片进行编码，转为byte类型的数组再用socket传送。

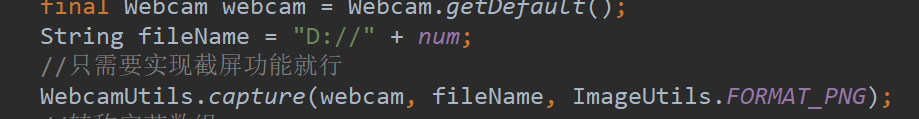
**2.5 摄像头监控具体实现**

**2.5.1 引入依赖：**

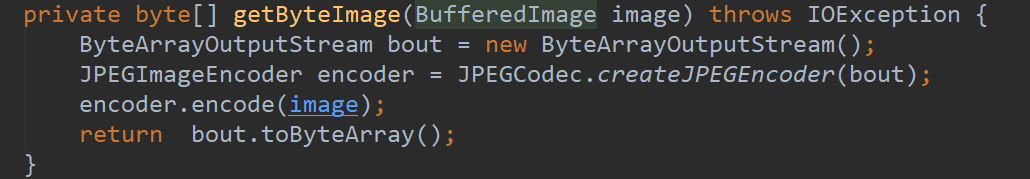


**2.5.2 实例化**

* 调用capture方法

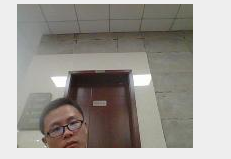


* 将图片转为byte类型数组的方法：



* 之后通过socket进行传输即可。

**2.5.3 效果**



**2.6 屏幕监控实现思路**

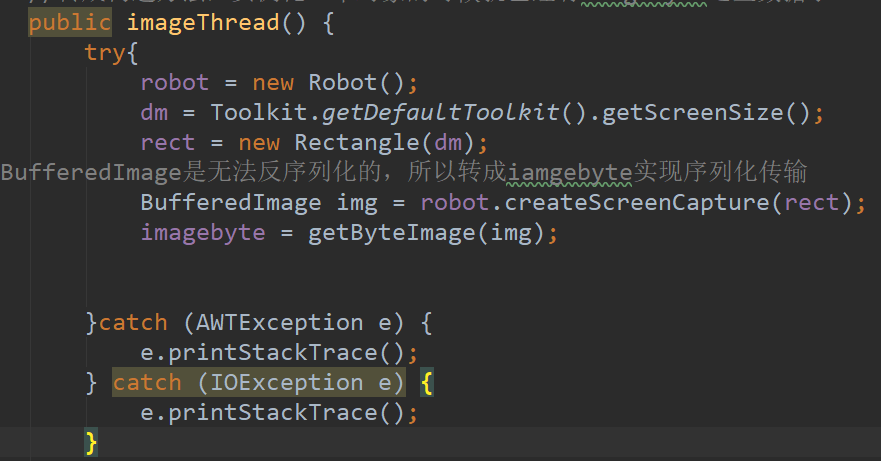
java有一个用来自动测试的robot类，顾名思义就是机器人，在实例化这个对象之后，可以通过调用creatScreenCapture方法来进行截屏。之后将图片进行编码传输到客户端进行解码即可。

**2.7 屏幕监控具体实现**

**2.7.1 实例化robot对象**

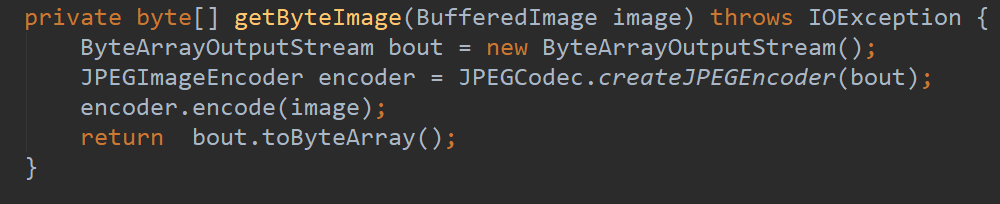
* 调用creatScreenCapture方法：

之所以要编码成byte数组类型，因为BufferedImage类是无法序列化的。

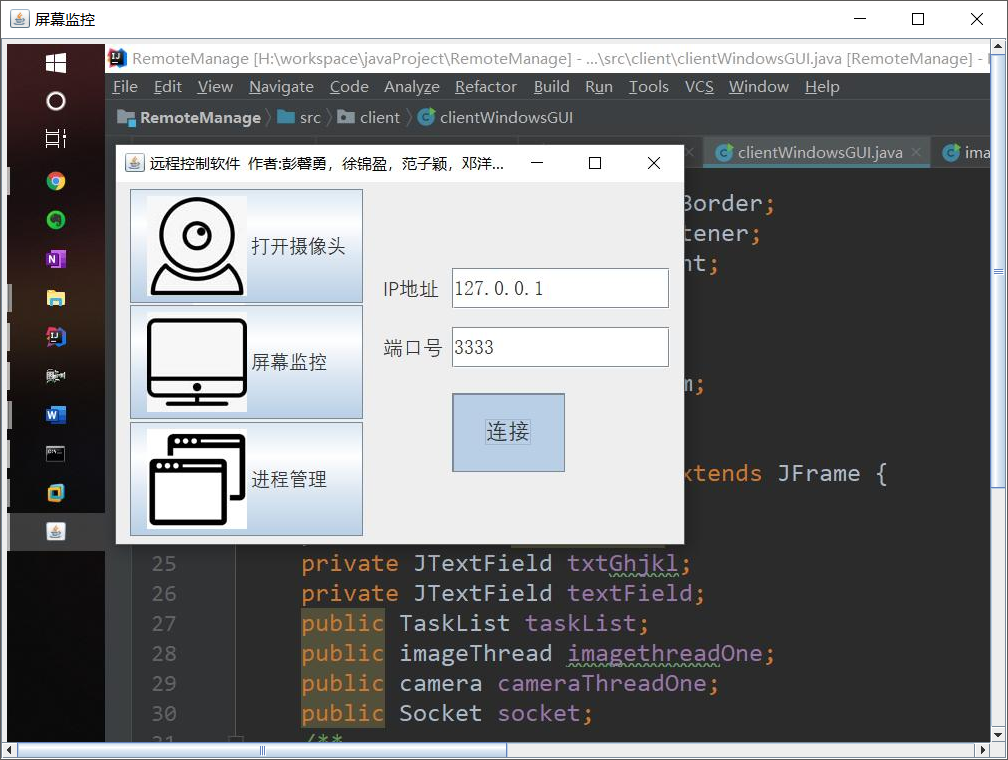


**2.7.2 图像编码**

* 实现如下：



**2.7.3 效果**



1. **防护软件设计**

* 制作的防护软件可考虑通过关闭程序所使用的端口来防止远程控制：

**import** **java.io.BufferedReader**;

**import** **java.io.IOException**;

**import** **java.io.InputStream**;

**import** **java.io.InputStreamReader**;

**import** **java.util.ArrayList**;

**import** **java.util.HashSet**;

**import** **java.util.List**;

**import** **java.util.Scanner**;

**import** **java.util.Set**;

**import** **java.util.regex.Matcher**;

**import** **java.util.regex.Pattern**;

**public** **class** **KillServer** {

**private** Set<Integer> ports;

**public** **static** void main(String[] args) **throws** InterruptedException {

System.out.println("请输入要杀掉的windows进程的端口号，如果有多个，则以逗号相隔");

System.out.println("Please input kill port");

Scanner scanner = **new** Scanner(System.in);

String input = scanner.next();

scanner.close();

String[] split = input.split(",");

Set<Integer> ports = **new** HashSet<>();

**for** (String spid : split) {

**try**{

int pid = Integer.parseInt(spid);

ports.add(pid);

}**catch**(Exception e){

System.out.println("错误的端口号，请输入一个或者多个端口，以英文逗号隔开");

**try** {

Thread.sleep(5000);

} **catch** (InterruptedException e1) {

e1.printStackTrace();

}

System.exit(0);

}

}

KillServer kill = **new** KillServer();

kill.ports = ports;

System.out.println("need kill " + ports.size() + " num");

**for** (Integer pid : ports) {

kill.start(pid);

}

System.out.println("清理完毕，程序即将退出");

System.out.println("SUCCESS");

Thread.sleep(5000);

System.exit(0);

}

**public** void start(int port){

Runtime runtime = Runtime.getRuntime();

**try** {

*//查找进程号*

Process p = runtime.exec("cmd /c netstat -ano | findstr \""+port+"\"");

InputStream inputStream = p.getInputStream();

List<String> read = read(inputStream, "UTF-8");

**if**(read.size() == 0){

System.out.println("找不到该端口的进程");

**try** {

Thread.sleep(6000);

System.exit(0);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}**else**{

**for** (String string : read) {

System.out.println(string);

}

System.out.println("找到"+read.size()+"个进程，正在准备清理");

kill(read);

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

*/\*\**

*\* 验证此行是否为指定的端口，因为 findstr命令会是把包含的找出来，例如查找80端口，但是会把8099查找出来*

*\* @param str*

*\* @return*

*\*/*

**private** boolean validPort(String str){

Pattern pattern = Pattern.compile("^ \*[a-zA-Z]+ +\\S+");

Matcher matcher = pattern.matcher(str);

matcher.find();

String find = matcher.group();

int spstart = find.lastIndexOf(":");

find = find.substring(spstart + 1);

int port = 0;

**try** {

port = Integer.parseInt(find);

} **catch** (NumberFormatException e) {

System.out.println("查找到错误的端口:" + find);

**return** **false**;

}

**if**(**this**.ports.contains(port)){

**return** **true**;

}**else**{

**return** **false**;

}

}

*/\*\**

*\* 更换为一个Set，去掉重复的pid值*

*\* @param data*

*\*/*

**public** void kill(List<String> data){

Set<Integer> pids = **new** HashSet<>();

**for** (String line : data) {

int offset = line.lastIndexOf(" ");

String spid = line.substring(offset);

spid = spid.replaceAll(" ", "");

int pid = 0;

**try** {

pid = Integer.parseInt(spid);

} **catch** (NumberFormatException e) {

System.out.println("获取的进程号错误:" + spid);

}

pids.add(pid);

}

killWithPid(pids);

}

*/\*\**

*\* 一次性杀除所有的端口*

*\* @param pids*

*\*/*

**public** void killWithPid(Set<Integer> pids){

**for** (Integer pid : pids) {

**try** {

Process process = Runtime.getRuntime().exec("taskkill /F /pid "+pid+"");

InputStream inputStream = process.getInputStream();

String txt = readTxt(inputStream, "GBK");

System.out.println(txt);

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**private** List<String> read(InputStream in,String charset) **throws** IOException{

List<String> data = **new** ArrayList<>();

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(in, charset));

String line;

**while**((line = reader.readLine()) != **null**){

boolean validPort = validPort(line);

**if**(validPort){

data.add(line);

}

}

reader.close();

**return** data;

}

**public** String readTxt(InputStream in,String charset) **throws** IOException{

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(in, charset));

StringBuffer sb = **new** StringBuffer();

String line;

**while**((line = reader.readLine()) != **null**){

sb.append(line);

}

reader.close();

**return** sb.toString();

}

}

* 完善1）使其绕过2）的方法：根据对应的时区的系统时间来确定使用的端口，这样每次使用的端口可以看作是随机的，无法通过固定的检测某一端口来防止远程控制。
* 继续完善2）使其继续拦截1）的方法：通过检测所有开放端口的敏感内容来确定有害端口并关闭使用该端口的进程。